Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Aplicació de SVM a la predicció de resultats acadèmics

MEMÒRIA

Autor: Roger Navas Solé
Director: Luis José Talavera Méndez

Convocatòria: Juny 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest és un projecte de *Educational Data Mining* que té per objectiu predir el rendiment acadèmic en el tercer quadrimestre del Grau en Enginyeria en Tecnologies Industrials. Es parteix d'un *data set* de 139 494 entrades que corresponen a un període de 7 anys i s'utilitzen *Support Vector Machines* per a resoldre sis problemes de classificació: predir els casos de suspens en les sis assignatures d'aquest tercer quadrimestre.

El llenguatge de programació és *Python*, els algorismes de modelatge són de la llibreria *Scikit-Learn* i les eines per al preprocessament són de la llibreria *Pandas*.

Sumari

1. Introducció

El present treball consisteix en un projecte de mineria de dades que té per a objectiu la predicció de notes dels alumnes del Grau en Enginyeria en Tecnologies Industrials. Conté tots els apartats d'aquest tipus de projecte segons la metodologia esdevinguda estàndard CRISP-DM —exceptuant el d'implementació, per raons que seran evidents. La família d'algorismes de modelatge utilitzada és la de les màquines de vector de suport (*Support Vector Machine* en anglès; SVM d'ara endavant). S'ha programat en llenguatge *Python*, i les llibreries principals del projecte han estat: *Pandas* per l'emmagatzematge i manipulació de dades, i *Scikit-Learn* per a l'aplicació d'algorismes SVM.

A continuació, en el punt 1.1, es defineixen amb més detall els objectius inicials del treball. El capítol 2 està dedicat a una sumaria explicació del les eines de la mineria de dades que ens són rellevants. D'aquesta manera, s'exposen les quatre fases de les què està format un projecte de mineria de dades (preprocessament, modelatge, validació i implementació) i es defineixen breument els conceptes que s'utilitzen a les tres primeres, que seran les que es duran a terme en el treball. Així, al punt 2.1 es troba l'aparell teòric corresponent al capítol 3, i als punts 2.2 i 2.3 el corresponent al capítol 4.

1.1. Objectius del projecte

En concret, l'objectiu principal del projecte és l'obtenció d'un model capaç de preveure les notes de les assignatures del tercer quadrimestre en funció dels resultats del primer any, i, més específicament, si es tracta d'aprovats o de suspensos. La ETSEIB ha facilitat aquestes dades, que són anònimes (no contenen cap informació personal que pogués permetre identificar als estudiants darrera dels expedients). Es disposa per tant de les notes de totes les convocatòries des de 2010 (any de creació del grau) fins al 2017 inclòs.

L'ús de SVM per a obtenir el model de predicció no es va decidir al llarg del procés, forma part dels objectiu inicials. D'altra banda, l'objectiu formatiu inherent a tot projecte té una importància particular en aquest treball, doncs els coneixements previs de —en general— *Data Mining* i *Machine Learning* no són tant amplis com ho podrien ser els d'altres disciplines que sí es tracten específicament en assignatures curriculars del grau[1]. En aquest sentit, un altre dels objectius era realitzar totes les etapes pròpies d'un projecte de *Data Mining*, que s'expliquen en el següent capítol.

[1] En un intercanvi Erasmus a l'escola d'enginyeria CentraleSupélec vaig tenir la oportunitat de cursar una assignatura optativa centrada en els models autoregressius, que tenen però pocs punts en comú amb els SVM (el preprocessament requerit i el funcionament general del algorisme són diferents).

1.2. *Data Mining:* definició i fases

La mineria de dades (*Data Mining* en anglès) és una disciplina de les ciències de la computació a l'alça: la medicina, l'economia o la sociologia són només alguns dels sabers que han integrat en els darrers anys aquest conjunt de tècniques a les seves recerques; la proliferació en el món empresarial ha sigut igualment ràpida, i l'alta demanda laboral de *Data Scientists* s'ha traduït al sector educatiu en una gran oferta de màsters universitaris dedicats íntegrament a aquesta disciplina. En l'àmbit de les enginyeries existeixen molts problemes que s'han optimitzat recentment amb l'ús de *Data Mining*; alguns exemples són: la predicció del punt de fallada d'estructures en enginyeria civil, el control dels corrents en instal·lacions d'alta tensió en enginyeria elèctrica, o la identificació de fraus mitjançant el tractament de dades telefòniques en enginyeria de les telecomunicacions [5].

No obstant, la mineria de dades no és una disciplina nova. En la cerca dels seus orígens ens podem arribar a remuntar als anys 1930, que és quan estan documentat els primers usos del concepte de *Knowledge Discovery in Databases* [1], encara que amb un sentit ben diferent del que tindrà als anys 90, quan prendrà importància. El *boom* recent de la mineria de dades està directament lligat al *boom* d'internet i a la "societat de la informació" que aquest ha portat amb si: l'ús generalitzat d'internet en tots els aspectes de la vida individual i col·lectiva genera una enorme quantitat d'informació, de dades que, per ser compreses i rendibilitzades, han de ser estudiades amb tècniques de *Data Mining*. Algunes altres raons que expliquen la popularitat de la mineria de dades són [1]: l'augment en la generació i en l'emmagatzematge de dades a nivell empresarial; la necessitat de disposar d'instruments per a prendre decisions cada cop més complexes; l'evolució general de la tecnologia, i dels sistemes de tractament de dades.

Si aquest augment en la disponibilitat de dades ha provocat un interès creixent en *Data Mining* és perquè aquest es pot definir com la ciència que estudia grans quantitats de dades (de l'ordre de Gigabytes). De fet, un projecte de mineria de dades es pot pensar com el tractament d'un *data set*, d'un conjunt de dades, mitjançant tècniques de *Machine Learning* [1]. Més rigorosament, la mineria de dades es pot definir com: "a collection of techniques for efficient automated discovery of previously unknown, valid, novel, useful and understandable patterns in large databases. The patterns must be actionable so they may be used in an enterprise's decision making" [1].

Observem que aquesta definició posa l'èmfasi en la descoberta de patrons vàlids en un conjunt de dades. En efecte, el *Data Mining*, al contrari que les ciències naturals tradicionals, no procedeix analíticament a partir d'unes determinades lleis per tal de trobar la solució d'un problema determinat, sinó que, en ordre invers, parteix del problema concret, del *data set*, per a trobar-hi una llei, algun tipus de patró o model que rarament té un sentit físic més enllà de

la correlació entre certes variables dimensionals.

Podem complementar aquesta primera definició amb una altra de caràcter més tècnic, que fa referència a les fases que formen un projecte de mineria de dades: “data science is the adaptive, iterative, and phased approach to the analysis of data, performed within a systematic framework, that uncovers optimal models, by assessing and accounting for the true costs of prediction errors” [11]. Notem que introdueix diversos matisos respecte a la primera: especifica que es tracta d’un estudi per fases de les dades, estudi que és *adaptatiu*, que troba el model *òptim*, i que ho fa —que verifica que és el model òptim— *valorant la imprecisió o error* dels models. Aquestes tres característiques que subratllem, i que la definició enuncia una rere l’altra, es corresponen a les tres fases que executarem en aquest treball: un projecte de mineria de dades ha d’adaptar en primer lloc la informació de què es disposa a un format adequat als algorismes de modelització (*fase de preprocés de dades*); un cop s’ha fet aquesta primera manipulació de les dades, es poden aplicar els algorismes, que donen sempre com a resultat el model òptim per a unes certes variables que depenen de cada algorisme concret (*fase de modelatge*); no obstant, en una tercera fase podem estimar l’error del model que hem obtingut i modificar conseqüentment aquestes variables del algorisme per a obtenir un model veritablement òptim dintre de la família d’algorismes utilitzada (*fase de validació*). Cal destacar un darrer detall de la definició: és un procés iteratiu, doncs els resultats d’una fase poden obligar a fer de nou les anteriors.

La divisió d’un projecte de mineria de dades en aquestes fases va ser establida formalment sota el nom de *Cross-Industry Standard Process for Data Mining* (CRISP-DM). Aquest estàndard va néixer al 1999 com a resultat de la col·laboració de grans empreses de sectors diferents, notablement Daimler-Benz (Automòbil), OHRA (Assegurances), NCR Corp. (Informàtica) i SPSS (Estadística) [7]. Aquest estàndard es presenta actualment amb lleugeres modificacions (veure per exemple [11]), però continua sent la referència. Com il·lustra el següent diagrama, la metodologia CRISP defineix de fet 6 fases. En efecte, a les tres fases que acabem de mencionar i que realitzarem en aquest projecte s’afegeixen tres fases més, dues anteriors i una posterior. Les dues fases prèvies són les fases *Business understanding* i *Data understanding*, que fan referència al procés d’entendre, d’una banda, el funcionament de l’empresa en la què s’està realitzant el projecte de mineria de dades i, de l’altra, el sentit i l’origen de les dades del projecte. La fase posterior és la fase *Deployment*, que fa referència al treball de seguiment necessari per la correcta implementació del projecte.

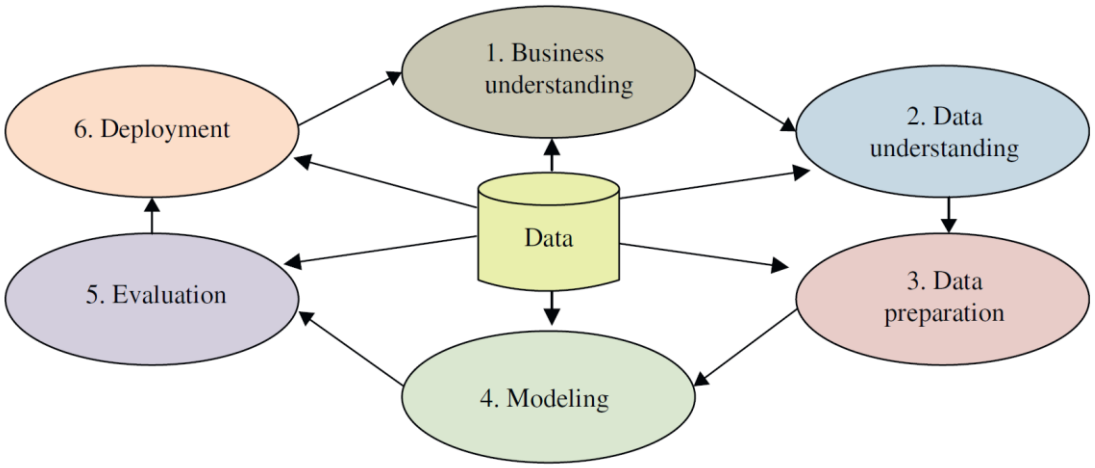


Figura 1: Fases establertes per la metodologia CRISP. Extret de [7].

No obstant, la primera i segona fases no requereixen una dedicació específica en el nostre cas, són trivials; la darrera fase tampoc és pertinent, doncs no té sentit fora d'un context d'empresa o de posada en pràctica efectiva del coneixement obtingut gràcies a la mineria de dades. En efecte, podem ometre les dues primeres fases perquè les tenim integrades pel fet d'haver sigut estudiants de les assignatures que estem tractant. Convé no infravalorar la importància de la segona fase, és a dir, d'entendre profundament la naturalesa de les dades objecte d'estudi; aquestes poden ser molt complexes i és imprescindible conèixer exactament el seu significat i el procés d'obtenció que hi ha al darrera; en el nostre cas, però, són relativament senzilles, i les consideracions que haguem de fer al respecte les inclourem en la fase de preprocés o *Data Preparation*.

2. Fonaments teòrics

En el primer dels tres apartats de què consta aquest capítol, el 2.1, farem unes breus consideracions sobre la primera de les tres fases del projecte, la fase de preprocessament de dades, que s'executa en el capítol 3. Els dos apartats següents estan dedicats, respectivament, a la segona i tercera fases, les fases de modelatge i validació, que s'executen conjuntament en el capítol 4. Així doncs, el segon apartat, el 2.2, explica els fonaments matemàtics del modelatge amb algorismes SVM, i el tercer i darrer apartat, el 2.3, les metodologies emprades per poder avaluar i validar qualsevol model de *Machine Learning*.

Aquest capítol pretén establir els fonaments teòrics d'un projecte de mineria de dades i, en concret, d'un projecte de modelització amb SVM; no s'hi trobaran consideracions sobre les particularitats de la implementació d'aquest tipus de projecte en *Python*, que es reserven pels capítols posteriors, on apareixeran en relació al projecte específic que ens ocupa.

2.1. Preprocés de dades

Aquesta fase sol ocupar el 90% de la duració total del projecte [1]. Podem dir per tant que la feina del *data scientist* consisteix principalment en la manipulació de les dades per tal de que puguin ser processades per un algorisme, doncs aquest és l'objectiu d'aquesta fase. En el nostre cas, però, aquesta regla general respecte la càrrega de treball no s'ha complert, doncs les dades que estudiàvem, encara que tenen una complexitat que fa possible i demana un projecte de *Data Mining*, no tenen les complicacions que produeixen aquesta divisió del temps en la majoria de projectes reals; i, d'altra banda, no teníem una formació en mineria de dades que fes de les següents fases un procés mecànic.

I quin és aquest format que s'ha de donar a les dades per permetre que siguin processades per un algorisme? Depèn de quin és l'objectiu del projecte i de l'algorisme que es vol utilitzar.

Els algorismes es classifiquen en dos tipus en funció del format de les dades que demanen. El algorismes de tipus *supervised learning*, com és el cas dels SVM que utilitzarem, requereixen tenir mostres de la sortida del model, en el nostre cas les notes de les assignatures del tercer quadrimestre. Així, les dades en divideixen en un conjunt *X* format per les dades d'entrada i un conjunt *Y* format per les dades de sortida. Es diu que el model associat a l'algorisme *s'entrena* amb aquests dos conjunts de dades disponibles, i es pot utilitzar posteriorment per a *predir* la sortida de dades que tinguin el mateix format que les dades d'entrada. Per contra, els algorismes del tipus *unsupervised learning* no requereixen disposar de mostres de la sortida del model; de fet, el seu objectiu no és necessàriament calcular una variable de sortida sinó trobar patrons en unes certes dades d'entrada.

En el nostre cas, les dades d'entrada, el conjunt *X*, seran les notes de les assignatures del primer any del grau. Ara bé, el conjunt *X* no estarà format estrictament pel resultats obtinguts a l'avaluació final: abans de passar a la fase de modelatge, manipularem les dades per obtenir altres variables que es deriven de les notes, com ara bé les vegades que s'ha suspès una assignatura.

Les dades de sortida, el conjunt *Y*, seran les notes de les assignatures del tercer quadrimestre del grau. Cal destacar que —a diferència de les notes del primer any— no conservarem la nota numèrica sobre 10, sinó que treballarem amb valors de "suspès" i "aprovat", doncs l'objectiu del projecte és obtenir un model capaç de predir i d'alertar de possibles suspensos. És a dir, treballem amb unes dades *Y categòriques o nominals* (per oposició a unes dades *Y numèriques o continues*): volem predir si un estudiant determinat aprovarà o suspendrà una assignatura, no si obtindrà un 4.7 sobre 10 o un 9 sobre 10. Com era el cas amb la distinció entre *supervised* i *unsupervised learning*, els algorismes de *Machine Learning* es poden classificar també en funció d'aquesta última dicotomia entre dades *Y* categòriques i dades *Y* continues.

En el capítol 3 d'aquesta memòria s'explica amb detall quin és el format exacte de les dades proveïdes per la ETSEIB i les operacions necessàries per a transformar-lo en un format apte per poder aplicar als algorismes SVM.

2.2. Modelatge

2.2.1. Tipus d'algorismes

Existeixen molts tipus d'algorismes de *Machine Learning*. Acabem de comentar en l'apartat anterior dos conceptes que serveixen per a fer-ne una taxonomia: el conjunt d'algorismes es divideix, d'una banda, en algorismes de *supervised learning* i algorsimes de *unsupervised learning* i, d'una altra banda, en algorismes que treballen amb dades numèriques i algorismes que ho fan amb dades categòriques. Creuant aquestes dues maneres de distingir els algorismes s'obté la següent taula, amb quatre tipus d'algorismes[2]:

[2] No hi ha una única manera de classificar els algorismes de *machine learning*; de fet, com argumentem tot seguit, és molt difícil establir fronteres nítides, ja que la regla general és que a una mateixa família d'algorismes pertanyin algorismes utilitzats per a diferents tipus de problemes. La taula que presentem és una versió modificada de l'original de la referencia bibliogràfica, que és discutible i no s'adequa al nostre context. La taula ens serveix simplement per introduir els diferents tipus de problemes amb els que tracta la mineria de dades a partir dels conceptes de *supervised-unsupervised* i de numèric-categòric.

	Dades numèriques	Dades categòriques
Supervised	Regressió	Classificació
Unsupervised	Reducció de dimensionalitat	*Clustering*

Taula 1: Tipus d'algorismes. Basat en [13].

Com ja hem avançat, ens trobem en el cas de *supervised learning* i de dades categòriques, i doncs en un problema de classificació. Cal notar que els algorismes SVM també poden ser de regressió; aquesta taula pretén ser un resum dels diferents tipus de problemes en el món de la *Data Science*; en una mateixa família d'algorismes —com els SVM— podem trobar algorismes de regressió i algorismes de classificació[3]. D'altra banda, dintre dels algorismes de *clustering* trobem molts mètodes per al tractament de dades numèriques.

Podem descriure molt breument cadascun dels quatre tipus d'algorismes, començant pels dos tipus de *unsupervised learning* que no utilitzarem en aquest projecte.

Els algorismes de reducció de dimensionalitat busquen una base de vectors propis que redueixi el rang del problema, és a dir, el nombre de dimensions o variables significatives. En el següent exemple gràfic si prenem la base $\vec{u} = \{u_1, u_2\}$ en comptes de la base original $\vec{x} = \{x_1, x_2\}$, podem considerar que les dades només depenen d'una dimensió o variable, u_1, en comptes de les dues variables de les què depenien inicialment.

[3] Els SVM poden arribar a ser algorismes amb característiques de *unsupervised learning*: són els algorismes coneguts com Semi-supervised Support Vector Machines o S3VM.

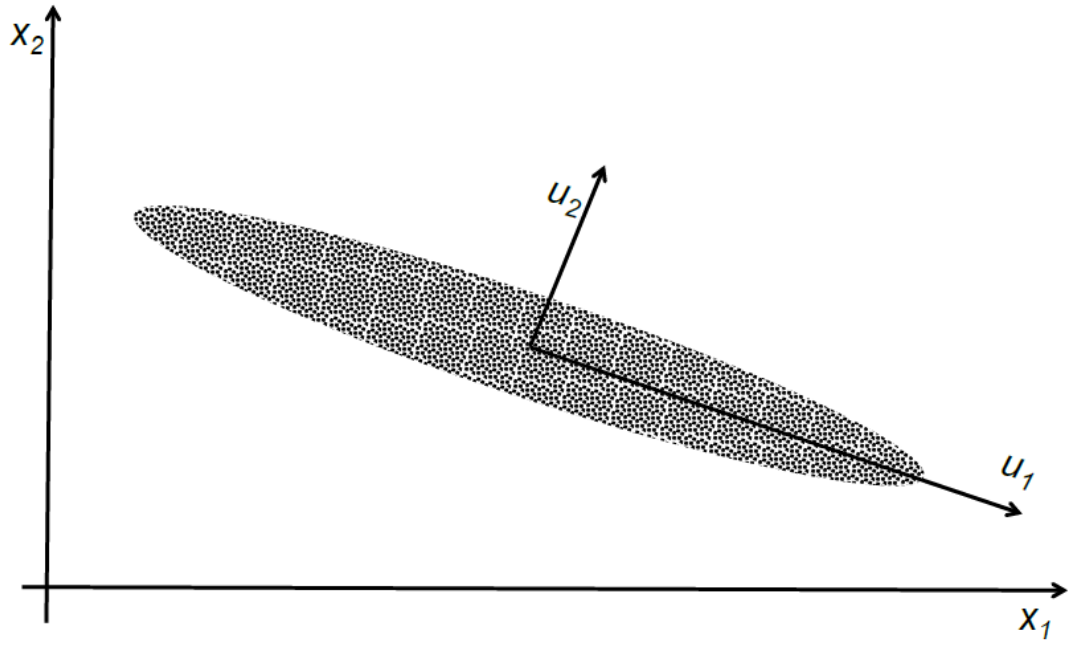


Figura 2: Representació d'un algorisme de reducció de dimensionalitat. Extret de [15].

Els algorismes de *clustering* agrupen les dades del problema en categories (categories generades pel propi algorisme que poden però tenir un sentit físic).

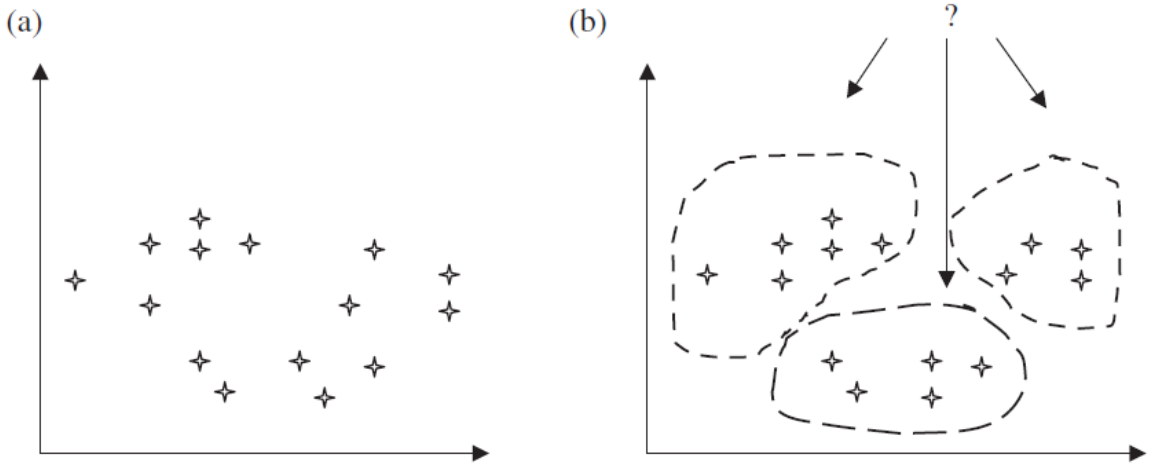


Figura 3: Representació dels algorismes de Clustering*. Extret de [8].*

Els algorismes de regressió —al igual que la regressió entesa fora del context de la mineria de dades— busquen predir respostes numèriques Y_i per unes dades d'entrada qualsevols X_i gràcies a la descoberta de les relacions entre uns conjunts de dades *X* i *Y* coneguts.

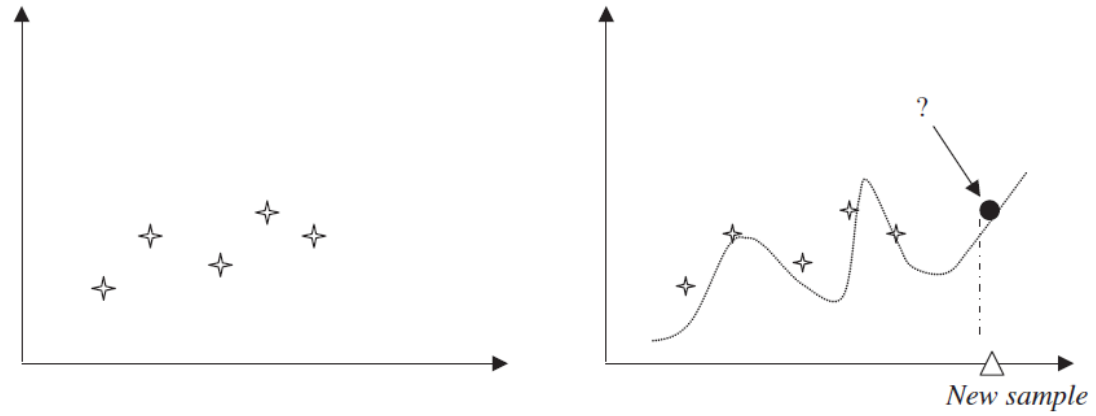


Figura 4: Representació dels algorismes de regressió. Extret de [8].

Els algorismes de classificació, com els que utilitzarem en el projecte, cerquen les fronteres que separen un tipus de dades d'uns altres tipus de dades (aprovats de suspensos en el nostre cas), de manera que son capaços de determinar el tipus d'unes noves dades X_i qualsevols (les notes de primer any d'un estudiant donat, en el nostre cas). Són algorismes que classifiquen per tant les dades en categories que —a diferència del que passa en els algorismes de *clustering*— s'han definit prèviament i no es generen automàticament com a conseqüència de l'execució de l'algorisme. L'algorisme s'entrena amb un conjunt de dades *X* de les quals es coneix a quines categories *Y* corresponen —per això és un algorsime de *supervised learning*. Com a resultat de l'entrenament s'obtenen les fronteres que separen les diferents categories, de manera que el tipus d'una nova dada qualsevol $\vec{x}_i = \{x_1, \dots, x_n\}$ es pot determinar molt ràpidament en funció de la regió de l'espai vectorial en què es troba.

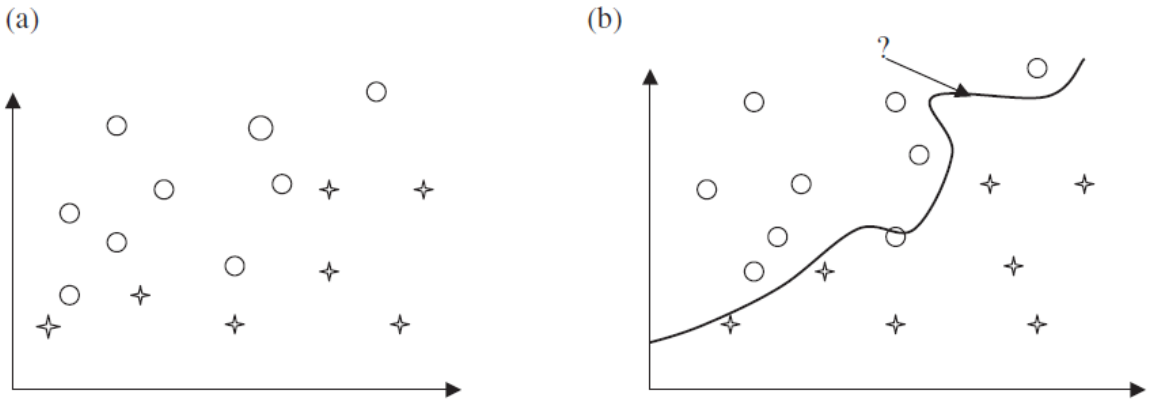


Figura 5: Representació dels algorsimes de classificació. Extret de [8].

Encara que el nostre projecte està centrat en l'aplicació d'algorismes SVM, utilitzarem també l'algorisme de la regressió logística. De fet, abans d'intentar obtenir un model per SVM, obtindrem un model per regressió logística que ens servirà de *baseline*, és a dir, de punt de referència i comparació per avaluar els models que construirem després per SVM. Els models

de regressió logística donen en la majoria dels casos pitjors resultats que els models de SVM [9], i per això podem utilitzar aquesta tècnica com a *baseline*: els resultats que obtinguem per regressió logística seran una cota mínima dels resultats que esperem obtenir per SVM.

2.2.2. La regressió logística

Els algorismes de regressió logística, encara que es basin en la tècnica estadística de la regressió, són algorismes de classificació —i no de regressió. Busquen predir el tipus o categoria associada a unes noves dades, no predir un valor numèric associat a aquestes dades.

Parlem de "categories" per l'analogia amb els algorismes de *clustering* que acabem de comentar i perquè és el terme més entenedor en l'explicació del funcionament del algorisme —que consisteix en buscar fronteres entre "categories" de dades. Però en la majoria de contextos aquestes "categories" corresponen a esdeveniments futurs que es volen predir (de manera que gràcies a l'algorisme sabem que, si unes dades es troben en una certa regió de l'espai, un esdeveniment donat ocorrerà). En el nostre marc aquest esdeveniment és el suspens en una determinada assignatura del tercer quadrimestre (tindrem un model per cadascuna de les assignatures del tercer quadrimestre).

Encara que una regressió logística es basi en les tècniques de regressió, no és senzillament una regressió lineal o polinòmica amb els resultats numèrics transformats en categories o esdeveniments. És a dir, i en el nostre cas, fer una regressió logística no és fer una regressió a l'ús i transformar els resultats superiors o iguals a 5 en aprovats i els resultats inferiors a 5 en suspensos.

El funcionament d'una regressió logística és en canvi el següent: es defineix p com la probabilitat que l'esdeveniment estudiat succeeixi, i es preveuen els valors de p amb les tècniques de regressió, de manera que en els casos en què s'obtingui $p > 0.5$ la sortida serà 1 (l'esdeveniment succeeix) i en el casos en què s'obtingui $p < 0.5$ la sortida serà 0 (l'esdeveniment no succeeix). Però la regressió que es realitza per preveure p és en concret la següent:

$$\log\left(\frac{p}{(1-p)}\right) = \alpha + \beta_1 \cdot X_{1j} + \beta_2 \cdot X_{2j} + \beta_3 \cdot X_{3j} + \cdots + \beta_n \cdot X_{nj} \quad \text{(eq. 1)}$$

És a dir, una regressió linear en que la variable estimada no és p (que podria prendre aleshores valors superiors a 1), o per exemple $\frac{p}{10}$, que en el nostre cas particular sí que

asseguraria a la pràctica que els valors de p estiguessin entre 0 i 1 (seria l'operació que comentàvem de transformar les notes superiors a 5 en valors "aprovat", 1, i les notes inferiors a 5 en valors "suspès", 0), sinó $\log\frac{p}{1-p}$, també notat a vegades com $\operatorname{logit} p$, que assegura en tots els casos, per a qualsevol problema, que p prendrà valors entre 0 i 1 [10].

2.2.3. Support Vector Machines: una breu historia[4]

Històricament, podem parlar de tres desenvolupaments teòrics que porten a la formulació dels algorismes SVM que s'utilitzen actualment. El primer d'ells va ocórrer a principis del anys 60. En el context d'un problema de classificació de dades en dos tipus diferents (que és el context en què naixeran els algorismes SVM), Vapnik i Lerner van proposar al 1962 que el hiperplà òptim és aquell que té un marge més gran. Com explicarem més endavant, aquesta proposició és certa; fins als anys 90 aquests autors van treballar en demostrar-la matemàticament. El segon desenvolupament és l'article de Boser, Guyon i Vapnik de 1992 [2], en el que formulaven un algorisme molt semblant ja als algorismes actuals, incorporant un *kernel* en la maximització del marge —definim també aquest concepte seguidament. El tercer i últim moment destacat en la formulació d'algorismes SVM és l'article de Cortes i Vapnik de 1995 [4], que demostrava que en certs casos particulars la proposició de hiperplà de màxim marge no es compleix, i proposava una solució alternativa amb un marge *soft* —veurem també aquest concepte. Aquesta solució alternativa és una generalització dels algorismes anteriors (un algorisme de marge *hard* és equivalent a un cert algorisme de marge *soft* amb uns paràmetres determinats), i per tant és la que s'implementa d'una manera o una altra en tots els algorismes SVM actuals.

Aquesta és la historia teòrica de la gènesis dels algorismes SVM. Pel que fa a la seva historia pràctica, cal dir que van fer-se populars ràpidament ja que, per a determinats problemes, donaven millors resultats que els algorismes de xarxes neuronals. Podem citar l'algorisme SVM desenvolupat per l'empresa de telecomunicacions AT&T pels tests MNIST com exemple d'un cas real que va contribuir enormement a la popularitat d'aquests algorismes.

Actualment, continuen sent algorismes molt populars, gràcies principalment a la seva capacitat d'obtenir bons resultats amb *data sets* relativament petits [9].

2.2.4. Support Vector Machines: l'aparell teòric (hard margin)

Els algorismes SVM es basen en buscar el hiperplà amb marges màxims. El hiperplà es defineix com la frontera que separa un tipus de dades de l'altre tipus de dades (ens situem en

[4] Aquest apartat està basat principalment en l'article de Xinhua Zhang a l'enciclopèdia Springer de *Data Mining* [16].

un problema de classificació en dos tipus, fet que no implica pèrdua de generalitat); i el marge es defineix aleshores com la distancia més petita entre un tipus de dades i el hiperplà, tenim doncs dos marges (un per cada tipus). En la següent figura veiem dos possibles hiperplans que separen dades de dos dimensions o variables (X i Y); el hiperplà de la dreta és millor que el de l'esquerra ja que té uns marges més amples.

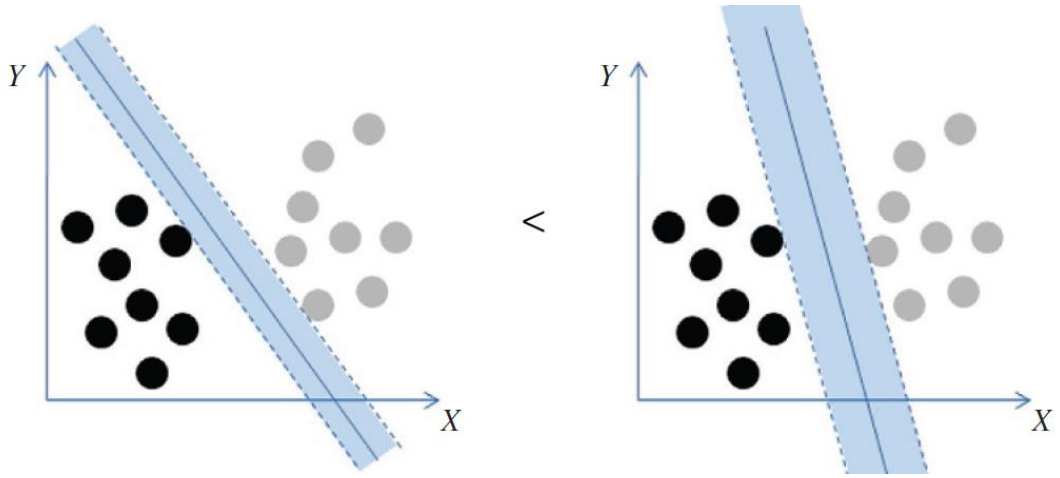


Figura 6: Dos hiperplans i els seus marges. Extret de [9].

No entrarem en detall en les demostracions matemàtiques sobre per què aquesta és una bona manera de separar els tipus de dades. Sí volem comentar les relacions entre l'amplada dels marges, la complexitat de la solució trobada per l'algorisme, i el principi SRM (*Structural Risk Minimization*), doncs seran importants en la fase d'avaluació.

El principi SRM és un principi general de *Machine Learning* que enuncia —simplificant— que tot algorisme ha d'intentar minimitzar la següent suma:

$$Error\ d'entrenament + K * Complexitat\ del\ model \quad \text{(eq. 2)}$$

Observem que la suma implica que no es pot minimitzar completament, alhora, l'error d'entrenament i la complexitat del model, hi ha un *trade-off* entre els dos, que es fixa en la constant K. L'error d'entrenament és l'error del model respecte a les dades amb les quals s'ha entrenat, les dades a partir de les quals formula el hiperplà. No ho justificarem, però —encara que pugui semblar estrany— aquest error és directament proporcional al marge. Si el marge és gran, l'error d'entrenament és gran, i aleshores, en base al *trade-off* del principi SRM, la complexitat del model pot ser menor. I això és el que volem, una complexitat del model petita, que evita els problemes d'*overfitting*, com argumentarem quan parlem de la fase d'avaluació.

Volem trobar, doncs, el hiperplà òptim que ens dona uns marges més grans. Tenim un conjunt de l punts x en una dimensió n que correspon al nombre de variables que prenem en

consideració en el nostre problema[5]. Si per exemple entrenéssim l'algorisme amb les 10 notes de primer any de 3000 estudiants, tindríem 3000 punts x de dimensió 10. En la bibliografia es parla sempre de vectors en comptes de punts, doncs és el que són matemàticament. Cada punt x està associat a una variable binaria y que ens indica el seu tipus, o, en el nostre cas, si l'esdeveniment associat succeeix o no, si suspèn o aprova una determinada assignatura de segon any; y pot valer 1 o -1. Matemàticament, notem el conjunt de vectors com:

$$D = \{(x_0, y_0), \dots, (x_{l-1}, y_{l-1})\}, \quad x \in \Re^n, \qquad y \in \{-1, 1\} \tag{eq. 3}$$

I el hiperplà el parametritzem com:

$$\langle w, x \rangle + b = 0 \tag{eq. 4}$$

És a dir, com el producte escalar entre un vector "pes" w i les variables lliures x de la dimensió en què ens trobem (no confondre amb els punts x_i), i la suma d'un paràmetre real b.

Tots els punts de cadascun dels dos tipus s'han de trobar a la banda adequada del hiperplà; el hiperplà ha de deixar tots els punts d'un tipus a una banda, i tots els punts de l'altre tipus a l'altra banda. Aquesta restricció es pot expressar matemàticament com:

$$y_i * (\langle w, x_i \rangle + b) \geq 1, \quad i = 0, \dots, l - 1 \tag{eq. 5}$$

D'altra banda, definim dos hiperplans més, paral·lels al hiperplà òptim; un d'ells a una banda de l'hiperplà òptim i l'altre a la banda oposada, és a dir, un a cada una de les dues regions del espai associades als dos tipus que volem classificar.

$$\pi_1: \langle w, x \rangle + b = 1 \tag{eq. 6}$$

$$\pi_2: \langle w, x \rangle + b = -1 \tag{eq. 7}$$

Es pot demostrar que la distància entre aquests dos plans és:

$$d = \frac{2}{\|w\|} \tag{eq. 8}$$

Aleshores, el problema que ens plantejàvem, trobar l'hiperplà que divideix les dades amb marge màxim, es tradueix en maximitzar la distancia d (els plans π_1 i π_2 no estan totalment definits i podem entendre que "es mouen") respectant la restricció (eq. 5). Hem reduït doncs el problema de trobar un algorisme que separi les dades amb marge màxim a un problema

[5] Seguim principalment l'explicació del manual de Mehmed Kantardzic [9], i utilitzem doncs, amb lleugeres modificacions, la seva notació.

d'optimització com els que s'estudien a l'assignatura de tercer any "Optimització i simulació". A partir d'aquí resolem aquest problema amb els mètodes que es van veure en aquesta assignatura

Observem en l'equació 8 que maximitzar d és equivalent a minimitzar $\|w\|$, que és equivalent al seu torn a minimitzar

$$\frac{\|w\|^2}{2} \quad \text{(eq. 9)}$$

Fem aleshores la langrangiana i obtenim que hem de minimitzar la funció

$$L(w,b,\alpha) = \frac{\|w\|^2}{2} - \sum_{i=0}^{l-1} \alpha_i * [\,(\langle w, x_i\rangle + b) * y_i - 1] \quad \text{(eq. 10)}$$

On α_i són els multiplicadors lagrangians. Computem el mínim d'aquesta funció (que té sempre mínim global) i aleshores els paràmetres que defineixen el nostre hiperplà òptim són:

$$w = \sum_{i=0}^{l-1} y_i x_i \alpha_i \quad \text{(eq. 11)}$$

$$b = -\frac{1}{2} w * (x_{tipus\,-1} + x_{tipus\,1}) \quad \text{(eq. 12)}$$

Molts dels multiplicadors lagrangians α_i acostumen a ser nuls. Els punts o vectors x_i que tenen associat un multiplicador lagrangià α_i no nul s'anomenen *support vectors* (d'aquí el nom de l'algorisme). $x_{tipus\,-1}$ i $x_{tipus\,1}$ són dos *support vectors* qualsevols que pertanyen respectivament al tipus o categoria codificada amb $y = -1$ i a la categoria codificada amb $y = 1$.

2.2.5. Support Vector Machines: l'aparell teòric (soft margin)

Hem formulat matemàticament la distancia d a maximitzar i la restricció de que el hiperplà deixi a una banda un tipus de dades i, a l'altra banda, l'altre tipus, i hem obtingut així un algorisme que troba el hiperplà òptim. En la gran majoria dels casos, però, és impossible trobar un pla que deixi *tots* els punts d'un tipus a una banda i *tots* els punts de l'altra tipus a l'altra. Hem de relaxar doncs la restricció i permetre que alguns punts es trobin en les regions equivocades. Es diu que en comptes de treballar amb un *hard margin*, treballem aleshores amb un *soft margin*.

Assumint doncs que tindrem punts en les regions que no els hi corresponen, el que fem és associar un cost a aquests punts i incorporar-lo a la funció a minimitzar (eq. 9), que queda

aleshores com

$$\frac{\|w\|^2}{2} + C * \sum_{i=0}^{l-1} \xi_i \qquad \text{(eq. 13)}$$

On C és una constant real arbitraria i ξ_i és la distància del punt x_i, que es troba a la regió inadequada, al pla més proper π_1 o π_2 que estem intentant fixar. D'altra banda, hem de modificar també la restricció (eq. 5), que ja no és absoluta i queda com:

$$y_i * (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \quad i = 0, \dots, l-1 \qquad \text{(eq. 14)}$$

Aquest problema d'optimització es resol anàlogament al problema amb *hard margin*.

La constant C, encara que sigui un paràmetre que escollim al aplicar l'algorisme, té també un sentit físic. Aquesta constant fixa de fet el que es defineix en rigor com el *soft margin*, que és —valgui la redundància— el marge més enllà del marge, és a dir, la distància màxima d'un punt al pla π_1 o π_2 —que definiria el *hard margin*— per la qual admetrem que estigui "mal classificat", que es trobi en una regió que no li correspon.

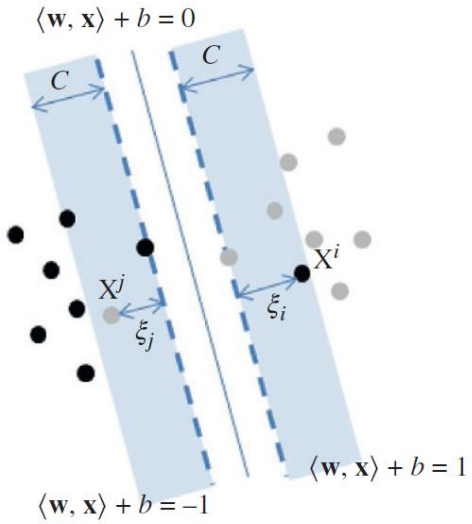


Figura 7: Visualització de C i ξ_i. Extret de [9].

Concretament, C és inversament proporcional a aquest *soft margin*, de manera que si C tendeix a infinit estem treballant en realitat amb un *hard margin*.

2.2.6. Support Vector Machines: l'aparell teòric (ús de *kernel*s)

Hem vist la manera en què un algorisme SVM classifica les dades, la manera en què estableix un hiperplà que les separa. Es podria pensar que els algorismes SVM només es poden aplicar

en problemes amb característiques lineals. Si la frontera que separarà els tipus de dades és un hiperplà, les dades han de presentar un comportament lineal; si no, necessitaríem geometries més complicades que un hiperplà per separar les d'un tipus de les de l'altre. No obstant, sempre podem linealitzar un problema: abans d'aplicar l'algorisme SVM podríem aplicar una funció a totes les dades, traslladar-les a un nou espai vectorial en el que tinguin un comportament lineal. Aquesta operació pot ser però computacionalment molt costosa, tant en termes de processament com en termes de memòria: hem d'aplicar una funció —que pot ser bastant complexa— a cadascuna de les dades amb les que entrenarem el model, duplicant-les a la pràctica, i, per fer prediccions, haurem de transformar també les dades d'entrada. Tanmateix, l'estructura de l'algorisme SVM fa que siguem capaços de realitzar aquesta operació amb uns costos molt menors. En efecte, podrem utilitzar una funció per linealitzar el problema sense haver de generar explícitament un nou espai vectorial, sense haver d'aplicar-la de forma bruta a totes les dades.

Això és possible ja que l'única operació que ha de fer l'algorisme amb les dades linealitzades és el producte escalar. Aleshores, en comptes d'aplicar primer una funció a totes les dades per linealitzar-les i després aplicar l'algorisme, podem en canvi definir una funció que calcula el que seria el producte escalar de les dades linealitzades, i podrem executar l'algorisme sense necessitat de linealitzar pròpiament les dades, reduint enormement els costos computacionals. En altres paraules, podem fer que l'algorisme treballi sobre unes dades no lineals modificant-lo lleugerament, substituint el producte escalar (que defineix el hiperplà, veure eq. 4) per una funció que calcula el valor del producte escalar en un altre espai vectorial. Aquesta funció s'anomena *kernel*, i té per definició la forma

$$K(x, y) = \langle \phi(x), \phi(y) \rangle \qquad \text{(eq. 15)}$$

On ϕ és la funció que altrament haguéssim aplicat a totes les dades per linealitzar-les.

En aquest projecte farem ús dels *kernels* predeterminats de la llibreria *scikit-learn*: el polinòmic, el "radial-basis function" (rbf) i el sigmoide. Tenen, respectivament, les següents tres expressions matemàtiques:

$$K(x, y) = (\langle x, y \rangle + c_0)^n \qquad \text{(eq. 16)}$$

$$K(x, y) = e^{-\gamma * \|x - y\|^2} \qquad \text{(eq. 17)}$$

$$K(x, y) = \tanh(\gamma \langle x^T, y \rangle + c_0) \qquad \text{(eq. 18)}$$

Fins i tot quan en rigor no apliquem una funció *kernel*, direm —segons és habitual— que tenim un *kernel* lineal.

Comentarem més en profunditat els tipus de hiperplans que s'obtenen amb cada *kernel* i la funció dels seus paràmetres en el capítol 4.

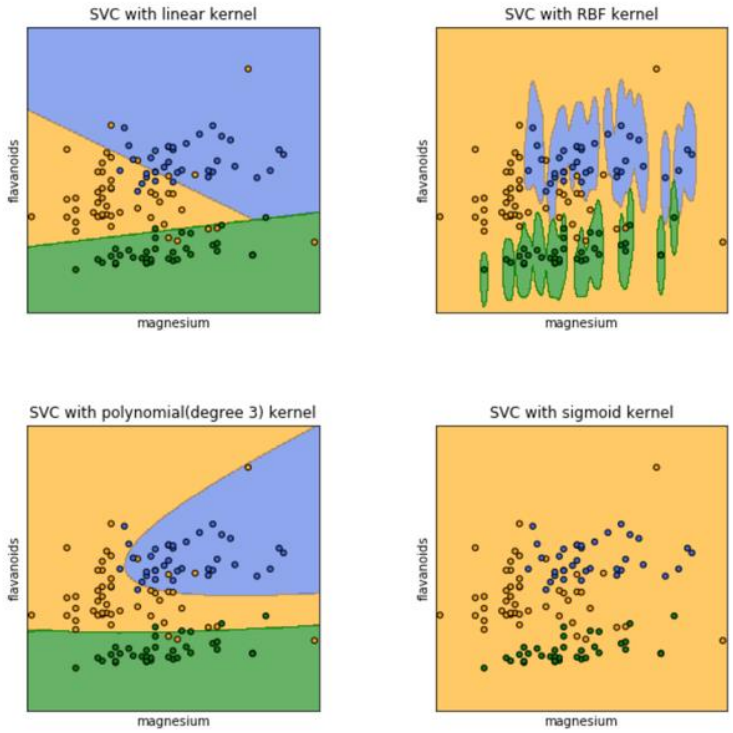


Figura 8: Resolucions d'un problema de classificació en 3 tipus amb els 4 kernels diferents que utilitzarem [17].

2.3. Validació del model

2.3.1. Objectius de la fase de validació

L'objectiu d'aquesta fase, la cinquena i penúltima segons la metodologia CRISP, és validar el model al que hem arribat a la fase anterior. Idealment, un model que ha estat validat correctament produirà bons resultats un cop comenci a utilitzar-se per resoldre el problema o problemes per als quals ha estat dissenyat, això és, un cop es passi a la fase de implementació, la darrera.

Per poder validar un model concret, cal ser capaç d'avaluar en general un model qualsevol: per tal de certificar que un cert model és aquell que ens donarà millors resultats, hem de tenir una metodologia que ens serveixi per avaluar, per puntuar tots els possibles models candidats, i ens permeti doncs seleccionar aquell que obtingui la millor puntuació.

Les fases de modelatge i de validació solen estar lligades. A la pràctica, es tracta d'un procés inductiu i no seqüencial: avaluem uns possibles models sobre els que estem treballant i,

segons els resultats que obtinguem, els desenvolupem més o els descartem. Per això ens interessa utilitzar l'algorisme de regressió logística com a *baseline*: si un cert model no ens dona millors resultats que els obtinguts per regressió logística, podem saber ràpidament que no cal treballar-hi més. A més, com hem vist a l'apartat 2.2.2, la regressió logística és un algorisme relativament senzill; ens servirà doncs per obtenir fàcilment una referencia a superar.

Per poder avaluar un determinat model analitzarem els resultats que dona per a unes certes dades *X*, per a les que coneixem les respostes reals *Y*. Farem que el model predigui uns resultats que de fet ja coneixem, i compararem els resultats previstos amb els resultats que s'haurien d'obtenir.

Sorgeix aquí un problema: com obtenim unes dades *X* amb respostes conegudes *Y*? Hem vist a l'apartat 2.1 que obtenim els models entrenant els algorismes amb, precisament, aquestes dades, les dades per a les que coneixem les respostes. En realitat, el que farem es dividir totes les dades de què disposem en dos conjunts, un *training set*, amb el qual entrenarem els algorismes, i un *test set*, amb el qual validarem els models. En altres paraules, ens reservem inicialment unes quantes dades per poder observar posteriorment el comportament del model i veure si encerta les prediccions. De fet, però, tampoc procedirem exactament d'aquesta manera. Aquesta operació de reservar dades és el que es coneix com *hold-out validation*, i és una opció vàlida, però en aquest projecte utilitzarem un altre mètode, anomenat *cross validation*, que té certs avantatges respecte l'operació de *hold-out* i que consisteix essencialment en repetir-la k vegades.

2.3.2. Obtenció d'un *test set*: mètodes de *cross validation*

Concretament, la *k-fold cross validation* consisteix en dividir les dades de què disposem en k *folds* o plecs i, de manera iterativa, utilitzar un d'ells com a *test set* i tota la resta com a *training set* fins que cadascun dels plecs ha servit de *test set*; cada iteració és independent de la resta, l'algorisme s'entrena "des de zero" cada vegada. Observis que si per exemple dividim en 5 plecs, tindre 5 iteracions, doncs cadascun dels plecs servirà una vegada de *test set*.

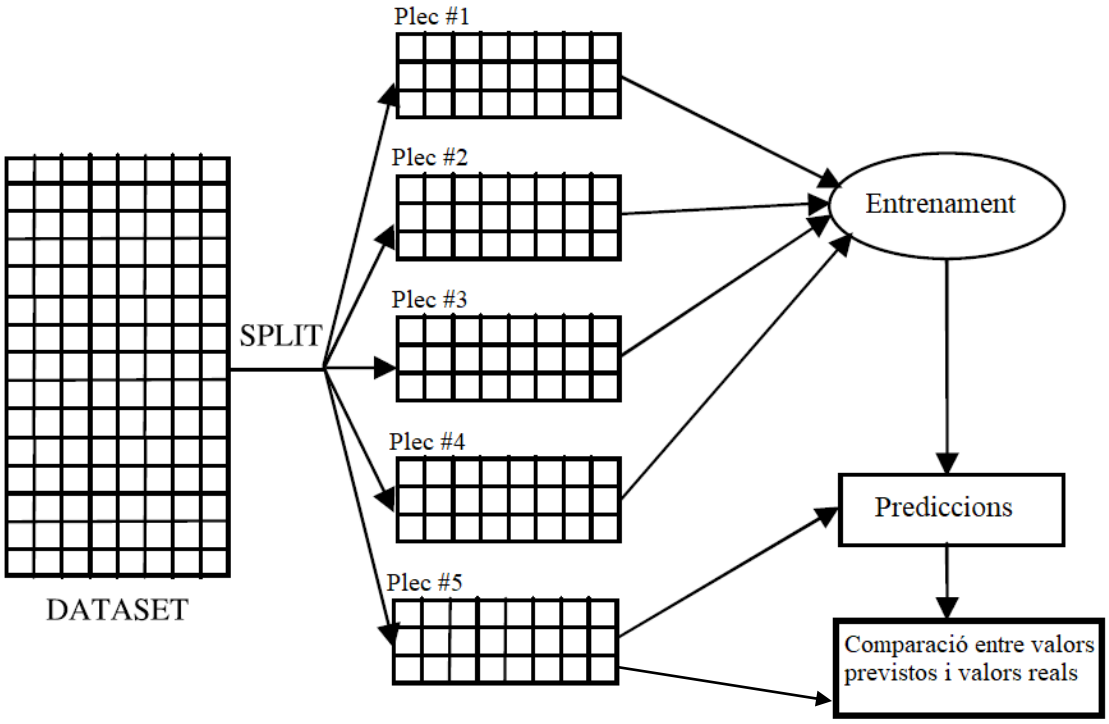


Figura 9: Darrera iteració d'una 5-fold cross validation*, en la què el cinquè i últim plec fa de* test set. *Basat en [3].*

És evident que el mètode de *cross validation* és computacionalment més costós que el de *hold-out*, doncs per exemple una *5-fold cross validation* equivaldrà aproximadament a cinc *hold-outs*. Però, d'altra banda, aquí rau precisament l'avantatge de la *cross validation*: per un mateix nombre de dades, farem un nombre molt més gran de comparacions entre valors previstos i valors reals. De fet, en la *cross validation* el nombre de comparacions és igual al nombre total de dades, doncs cadascuna de les dades forma part en una determinada iteració del plec que s'utilitza com a *training set*. Aquesta característica de la *cross validation* fa que sigui el mètode preferit en els casos en què no es disposa d'un nombre molt gran de dades, com és el nostre, doncs al augmentar el nombre de comparacions, la mida efectiva del *test set*, es redueix el risc de *overfitting*, és a dir, de tenir un model que separa perfectament les dades amb les quals s'ha entrenat però que dona en canvi mals resultats per a noves dades

2.3.3. Mètriques d'avaluació

Podem comparar de moltes maneres diferents els valors previstos i els valors reals. Les mètriques són els indicadors del rendiment d'un algorisme, i neixen de les diferents maneres en què podem fer aquesta comparació. Per avaluar els models i seleccionar el millor analitzarem els valors de cada model en una sèrie de mètriques.

Tant el valor previst com el valor real són variables binaries.

$$\hat{y}_i = \{0,1\}$$

$$y_i = \{0,1\}$$

Per cada resposta y_i tenim doncs quatre combinacions possibles.

	$\hat{y}_i = 0$ (aprovat previst)	$\hat{y}_i = 1$ (suspès previst)
$y_i = 0$ (aprovat real)	*True Negative* (TN)	*False Positive* (FP)
$y_i = 1$ (suspès real)	*False Negative* (FN)	*True Positive* (TP)

Taula 2: Combinacions possibles de valors reals i valors previstos

Les mètriques es defineixen a partir del nombre d'ocurrències d'aquestes quatre possibilitats. Aquests quatre valors amb els quals es calculen totes les mètriques es poden disposar en una matriu $2 * 2$ anàloga a la taula 2 i coneguda com la matriu de confusions del model.

Cada mètrica ens donarà informació diferent sobre el comportament del model. La mètrica *accuracy* es defineix com el nombre de previsions correctes dividit entre el total de dades.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad \text{(eq. 19)}$$

La mètrica *precision*, també coneguda com *positive predictive value*[6], és la proporció de resultats positius (previstos) que són realment positius.

$$Precision = \frac{TP}{TP + FP} \quad \text{(eq. 20)}$$

La mètrica *recall*, també coneguda com *true positive rate*, *sensitivity* i *hit rate*, és la proporció de positius reals que han sigut correctament classificats, previstos com tals, pel model.

$$Recall = \frac{TP}{TP + FN} \quad \text{(eq. 21)}$$

La mètrica F_1 és la mitja harmònica de les mètriques *precision* i *recall*.

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \quad \text{(eq. 22)}$$

[6] Els noms alternatius de les mètriques, així com la manera concreta de formular les definicions —que fàcilment pot portar a confusions—, els hem extret de [14].

Explicarem més en detall el sentit de les mètriques i quines prioritzem en l'apartat dedicat pròpiament a la realització de la fase de validació.

3. Preprocessament de dades

El capítol següent correspon a l'execució de la primera fase del projecte, la fase de preprocessament; s'hi explica doncs les operacions necessàries per transformar les dades proveïdes per la ETSEIB en dades aptes per a l'aplicació dels algorismes SVM

3.1. Objectius de la fase: format desitjat de les dades

Encara que existeixen models SVM que poden treballar amb valors "NaN", valors buits, indeterminats, els algorismes de *Scikit-Learn* que utilitzarem no els admeten. El preprocessament ha d'assegurar doncs que, siguin quines siguin les operacions que es fan i les dades que s'acaben introduint a l'algorisme, aquestes siguin completes, que existeixi un valor de cada variable per a cada entrada. L'eliminació de les dades "NaN", així com d'altres dades que s'identifiquin com corruptes, es coneix com *data cleaning*.

El format de les dades per poder aplicar SVM ha de ser doncs el següent:

Expedient	X_1	X_2	...	X_n	Y_1
Expedient #1	valor	valor	...	valor	valor
...	...	...	...	...	...
Expedient #m	valor	valor	...	valor	valor

Figura 10: Format desitjat de les dades.

Notem en primer lloc que només hi ha una variable de sortida Y_1. En efecte, els algorismes SVM només admeten una variable de sortida, i per tant haurem de construir un model diferent per a cada assignatura del tercer quadrimestre. Per contra, admeten un nombre enter *n* de variables d'entrada. Observem que les dades s'estructuren per expedients: totes les variables són valors en relació als expedients. Aquesta estructura es deriva del fet que l'objectiu del projecte és obtenir un model que, per a un expedient donat que hagi superat el primer any, sigui capaç de predir el rendiment a l'inici del segon any.

3.2. Les dades facilitades per la ETSEIB

La ETSEIB va proveir les dades acadèmiques en tres fitxers tipus excel amb noms "qfaseini", "qfasenoini" i "dadespernomespreins".

El darrer dels tres conté la informació generada durant el procés de preinscripció dels

estudiants. Aquesta informació és: l'any i la nota de les proves d'accés a la universitat (PAU), el codi postal del centre d'educació secundària, el codi postal del lloc de residència del estudiant, i el sexe del estudiant. És va decidir no utilitzar aquestes dades. Les raons són varies.

Sense entrar en debats sobre el sistema educatiu i el grau de correlació entre el rendiment acadèmic a la secundària i a la universitat, podem afirmar que la informació aportada per les notes del primer any a la universitat serà més rellevant que l'aportada per la nota de les PAU: d'una banda, és homogènia respecte a les variables que es volen predir, les notes en el segon any (doncs és informació provinent d'un mateix entorn, la ETSEIB) i, d'altra banda, és molt més rica (tenim 10 assignatures en comptes d'un examen). Les notes de les PAU podrien ser perfectament una de les variables X_i del model, però senzillament es va decidir no incloure-les, doncs no millorarien substancialment els models; si en comptes d'estudiar el rendiment acadèmic en el 3r quadrimestre del grau, estudiéssim el rendiment en el 2n quadimestre, sí que les incorporaríem, doncs no disposaríem de tanta informació.

Tampoc es va decidir prendre en compte el sexe del/la estudiant. La introducció de la perspectiva de gènere en la mineria de dades és actualment un important tema de debat (veure per exemple [12]); no és en cap cas un problema senzill que es resoldria simplement considerant una variable X_i definida com el sexe dels subjectes d'estudi.

D'altra banda, podríem creuar les dades sobre els codis postals amb bases de dades socioeconòmiques. De nou però estem tocant qüestions pròpies de la sociologia que són sovint menys evidents del que podria semblar. A més, no existeix una base dades que contingui rentes en funció dels codis postals, l'hauríem de generar a partir de dades amb altres divisions territorials.

Els altres dos fitxers excel, "qfaseini" i "qfasenoini", contenen les notes de les assignatures de la fase inicial (que correspon al primer any del grau) i les de la fase no inicial (que correspon als altres tres anys del grau) respectivament. Els dos fitxers tenen el següent format:

CODI_PROGRAMA	CODI_EXPEDIENT	CODI_UPC_UD	CREDITS	CURS	QUAD	SUPERA	NOTA_PROF	NOTA_NUM_AVAL	NOTA_NUM_DEF	GRUP_CLASSE
752	239833	240025	7.5	2010	2	N	1.4	1.4	1.4	52
752	239834	240025	7.5	2010	2	S	6.5	6.5	6.5	52
752	227908	240025	7.5	2010	2	N	2.6	2.6	2.6	43
752	228695	240025	7.5	2010	2	N	1.4	1.4	1.4	43
752	227332	240025	7.5	2010	2	S	7.7	7.7	7.7	51

Taula 3: Mostra de la taula de dades proveïda per la ETSEIB.

Observem que la taula conté altres dades apart de la nota. De fet, podem veure que conté tres notes diferents: "NOTA_PROF", "NOTA_NUM_AVAL" i "NOTA_NUM_DEF". A continuació s'explica quina és la informació continguda en cadascuna de les 11 columnes que composen la taula:

- CODI_PROGRAMA: fa referència al pla d'estudis. Ens interessaran únicament aquelles entrades (files) amb codi 752, que correspon al Grau en Enginyeria en Tecnologies Industrials.
- CODI_EXPEDIENT: indica l'expedient al que correspon l'entrada.
- CODI_UPC_UD: fa referència a l'assignatura a la que correspon l'entrada.
- CREDITS: indica el nombre de crèdits de l'assignatura.
- CURS: indica l'any en què s'ha cursat l'assignatura.
- QUAD: indica el quadrimestre en què s'ha cursat l'assignatura.
- SUPERA: indica si, en la convocatòria de l'entrada, s'ha superat l'assignatura.
- NOTA_PROF: indica la nota proposada pel professor.
- NOTA_NUM_AVAL: indica la nota un cop feta la primera avaluació.
- NOTA_NUM_DEF: indica la nota final de la convocatòria de l'entrada.
- GRUP_CLASSE: indica el grup al que estava matricula l'alumne en la convocatòria de l'entrada.

Cal destacar que, com ens indica el programa excel a la taula 1 amb un triangle a la cantonada superior esquerra de les celes, hi ha tres columnes (CODI_UPC_UD, QUAD i GRUP_CLASSE) que contenen valors numèrics en format text, *string*; ho haurem de tenir compte al manipular-los.

Utilitzarem la llibreria *Pandas* per emmagatzemar i manipular les dades. És obligat, doncs, transformar aquestes dades proveïdes per la ETSEIB en format excel en un objecte d'aquesta llibreria anomenat *Data Frame* (DF d'ara endavant), que és l'objecte utilitzat per emmagatzemar taules de valors. Ho fem mitjançant la següent comanda:

```
#Carreguem dades Fase Inicial
FaseIni=pd.read_excel('/Users/rn/Google Drive/2019-20/TFG Etseib/Gener/qfaseini.xlsx', sheet_name='Dades_Solano_qualif')
```

3.3. *Data cleaning*: filtrat de dades no rellevants

Abans de fer qualsevol operació amb les dades, convé eliminar aquelles que per alguna raó o altra hem de descartar; així, ens estalviem operacions innecessàries. Examinant el fitxer excel original observem que en efecte hi ha entrades que no contenen la nota real d'una convocatòria vàlida. Tenim 53.758 entrades en el fitxer "qfaseini" i 85.736 en el "qfasenoini", així que, com en tot projecte de mineria de dades, és impracticable detectar tots els valors anòmals a ull. Recorrem doncs a la funció de Pandas "duplicated" per a identificar els valors únics, no duplicats, i detectar així els valors estranys.

Observem així que en certes entrades: el valor de "GRUP_CLASSE" és la paraula "CONV"; el valor de "QUAD" és 0, no 1 —que correspon al quadrimestre de tardor— ni 2 —que correspon al de primavera—; no hi ha valors en les tres columnes de notes. D'altra banda,

haurem de descartar també totes les entrades que tinguin un valor de "CODI_PROGRAMA" diferent de 752, que és el que correspon al Grau en Enginyeria en Tecnologies Industrials.

Per eliminar tots aquests valors ens beneficiem d'una propietat dels DF que permet accedir de manera eficient a una fila en funció dels valors de les columnes. Utilitzem també la funció de *Pandas* "dropna" que elimina totes aquelles entrades que continguin alguna columna amb valor "NaN". Eliminem d'aquesta manera moltes entrades que no ens són de cap valor.

3.4. Modificar el format de les dades: *Pivoting*

Hem vist que a les dades facilitades per la ETSEIB a cada fila correspon una convocatòria i un alumne. Observem que aquest no és el format requerit per poder aplicar algorismes SVM, format que hem presentat a la figura 2. El canvi d'un format a l'altre és una operació habitual en els processos de manipulació de dades, i es coneix amb el nom de *pivoting*.

En efecte, és freqüent desar valors que són de tipus semblant però no idèntic (en aquest cas, les notes de cada estudiant individual) en una mateixa taula i utilitzar una variable anomenada categòrica[7] per distingir cada tipus particular (en aquest cas, la columna "CODI_EXPEDIENT"); és un format compacte i fàcil d'expandir, ja que evita tenir que crear una taula per a cada tipus particular de valor (en aquest cas, evita tenir una taula per cada estudiant). O, encara que no s'utilitzi aquest format concret que s'ha descrit, si més no és el format que s'obté en molts casos al demanar a un sistema d'emmagatzematge de dades unes dades concretes. Per contra, aquest no és un bon format per manipular ni per visualitzar les dades. *Pandas* disposa d'una funció ("pivot") que pren la variable categòrica com argument i l'utilitza per separar cada tipus particular de variable en una fila diferent (en aquest cas, el tipus ve donat per "CODI_EXPEDIENT", que ens dona informació categòrica sobre les notes, especificant l'estudiant a qui corresponent). Aquest argument rep lògicament el nom de "index".

La funció "pivot", doncs, pren una columna, el "index" i, per cada valor diferent que contingui aquesta columna, crea una nova fila. Els valors d'aquestes noves files seran els valors d'una o més columnes del DF original especificades amb l'argument "values".

Podem en efecte tractar més d'una columna alhora. Ho haurem de fer en els casos en què tinguem més d'una variable categòrica, com és el nostre: una nota a la columna "NOTA_NUM_DEF" correspon *a l'estudiant especificat a "CODI_EXPEDIENT" i a una assignatura especificada a "CODI_UPC_UD"*. Si recordem el format desitjat de les dades a la figura 2, veurem que necessitàvem estructurar les dades de manera que a cada fila

[7] Utilitzem els termes de l'assignatura de Tècniques estadístiques per a la qualitat.

correspongui un estudiant però que podíem tenir més d'una variable X_i per cada estudiant. De fet ens interessarà tenir, segur, les notes de totes les seves assignatures. L'argument "columns" de la funció "pivot" ens permet obtenir-les; en ell, s'introdueixen les variable categòriques secundàries supeditades a la variable categòrica primària utilitzada com a "index" (en el nostre cas, "CODI_UPC_UD", que es supedita a "CODI_EXPEDIENT"). Finalment, el darrer argument de la funció "pivot", "values", que serveix per a indicar a la funció en quina columna es troben els valors als que fan referència les variables categòriques, serà en el nostre cas "NOTA_NUM_DEF". La següent figura, de la documentació oficial de *Pandas*, il·lustra l'operació:

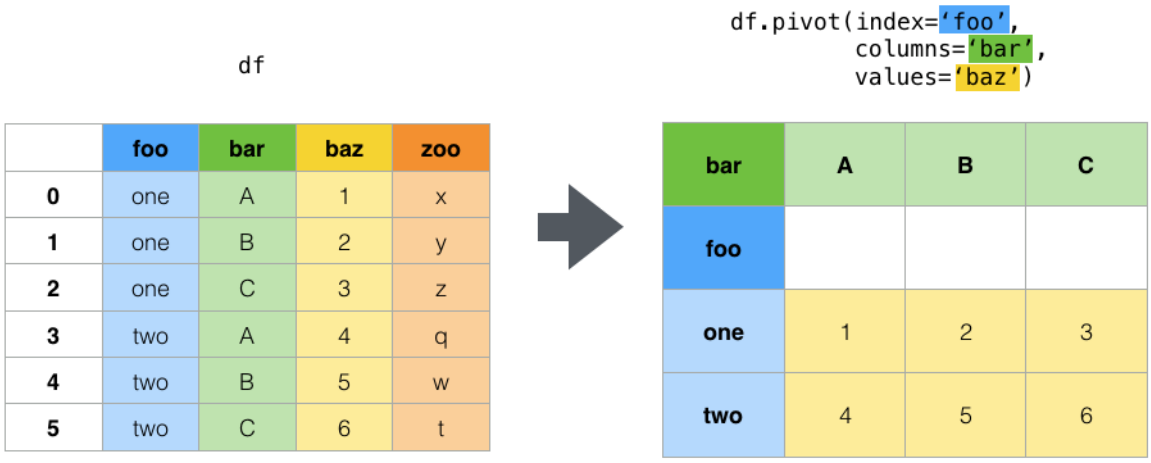


Figura 11: Operació de pivoting. *Extret de [17].*

Prenem "NOTA_NUM_DEF" ja que, entre les tres notes diferents de la taula original, és la nota final de l'estudiant i doncs la més significativa.

Abans de fer l'operació de *pivoting*, però, podem fer operacions amb les dades per obtenir altres variables predictores, altres X_i que introduirem també com arguments "values" a la funció *pivoting*, doncs seran variables que, al igual que les notes, faran també referència *a un* estudiant i *a una* assignatura. En concret, vam considerar que arribaríem a un millor model si teníem en compte quina va ser la mitja en la convocatòria en què l'estudiant va aprovar l'assignatura i quantes vegades es va matricular l'estudiant en aquella assignatura, o, el que és el mateix, quantes vegades l'havia suspesa anteriorment.

3.5. Càlcul d'altres variables predictores.

Volem, doncs, afegir dues columnes a la taula original.

La primera d'elles, que anomenem "%Mitja", la definim com la nota expressada en percentatge respecte a la mitja de la convocatòria. És a dir,

$$\%Mitja = \frac{NOTA\_NUM\_DEF}{Mitja\ Conv.} * 100 \quad \text{(eq. 23)}$$

Per a poder computar-la, necessitem primer una manera de computar la mitja de la convocatòria. Podríem iterar el DF per assignatura, any i quadrimestre per a calcular la mitja de cada convocatòria; en cada iteració accediríem (utilitzant la manera típica d'accedir a dades d'un DF en funció dels valors de les celes) a totes les notes d'una assignatura donada en un quadrimestre donat; i utilitzaríem la funció "mean" de *Pandas* per a calcular la mitja. *Pandas* té però una funció capaç de fer aquest càlcul de forma molt més eficient.

Es tracta de la funció "groupby" que, com el seu nom indica, agrupa totes les files que comparteixin el mateix valor en una o més columnes especificades. En el nostre cas, volem agrupar el valors de totes les files de "NOTA_NUM_DEF" que comparteixin any, quadrimestre i assignatura (és a dir, "CURS", QUAD" i "CODI_UPC_UD"). Pot semblar que la diferència entre accedir iterativament a les dades filtrant per any, quadrimestre i assignatura, i l'ús d'aquesta funció —que filtra també per aquestes categories— no sigui molt gran, però, com justificarem computant el temps de càlcul d'un i altre mètode, sí ho és.

La funció "groupby" permet triar la manera d'agrupar les files, i una de les opcions és fent la mitja amb la mateixa funció "mean". Obtenim així un DF amb les mitges de cada convocatòria.

Ens falta però afegir la columna en si, és a dir, fer el càlcul de l'equació 23 per a cada fila. Utilitzem per aquest propòsit el mètode de *Pandas* "apply", que permet aplicar a un DF per files una funció a definir. Aquest mètode passa sempre com a argument (a la funció que es defineixi) la fila individual del DF. Definim doncs una funció "PercMitja" amb dos arguments: la fila del DF original i el DF que hem obtingut amb l'ús de "groupby"; aquesta funció pot retornar aleshores el càlcul de l'equació 23. Així, amb només una línia de codi (i la funció "PercMitja") creem una nova columna en el DF original.

```
#Afegeixo columna %Mitja que es el tant per cent de la nota del alumne respecte a la mitja
FaseIni['%Mitja']=FaseIni.apply(lambda x: PercMitja(x, MitgesConv), axis=1)
```

El problema de computar el nombre de vegades que un estudiant s'ha presentat a una assignatura, el nombre de repeticions de l'assignatura, és molt semblant a l'anterior. En comptes de filtrar per any, quadrimestre i assignatura per a obtenir tots els estudiants que es van presentar en una mateixa convocatòria, volem ara filtrar per expedient i assignatura per a obtenir totes les vegades en que un estudiant es va presentar a la mateixa assignatura. De nou, podríem fer-ho utilitzant iterativament la comanda que permet accedir a dades d'un DF en funció dels valors de les celes, però és més eficient fer-ho amb el mètode "groupby". Notem, però, que la iteració en aquest cas es complica, doncs, a diferència del cas de la mitja, en què sabem abans d'iterar que busquem celes amb valors d'any que van del 2010 al 2017

i amb altres valors coneguts per a les columnes de quadrimestre i assignatura, en aquest cas no tenim manera de saber els valors de les celes a les que volem accedir. Iterar tot un DF és una operació que es sol evitar en mineria de dades, doncs és molt ineficient. Podem però utilitzar la funció de *Pandas* "duplicated" per a obtenir un DF que contingui només les files amb valors repetits d'expedient i assignatura, és a dir, les files d'assignatures que no van ser aprovades a la primera convocatòria; i iterem aquest DF (amb detalls en els que no cal entrar, però que asseguren el mínim nombre d'iteracions possible a l'interior d'aquest DF).

Utilitzant "groupby", però, podem fer la mateixa operació de manera molt més eficient. La manera d'agrupar les files no serà com abans "mean" sinó "size", que ens retorna el nombre d'ocurrències. Obtenim així un DF amb tres columnes: l'expedient, l'assignatura, i el nombre de vegades en què s'ha matriculat l'assignatura.

```
Repeticions=FaseIni[['CODI_EXPEDIENT', 'CODI_UPC_UD',]].groupby(['CODI_EXPEDIENT', 'CODI_UPC_UD',], as_index=False).size()
```

Ens falta però ajuntar-lo amb el DF que estem construint. Per a fer-ho utilitzem la funció de *Pandas* "merge" que permet unir dos DF que comparteixin valors en columnes de variables categòriques. Unim doncs els dos DF especificant que les columnes d'expedient i d'assignatura ("CODI_EXPEDIENT" i "CODI_UPC_UD") són les que es repeteixen en un i altre.

```
FaseIni=pd.merge(FaseIni, Repeticions, how='left', on=['CODI_EXPEDIENT', 'CODI_UPC_UD'])
```

Hem obtingut així un DF amb les dues columnes addicionals que volíem, "%Mitja" i el nombre de convocatòries a una assignatura, que anomenem "Rep".

Podem comparar els temps d'execució per a computar aquestes dues columnes en funció de si utilitzem "groupby" o si, de manera molt menys eficient, iterem. Per a fer-ho, utilitzem la funció "process_time" de la llibreria *time*. En comptes de buscar directament els temps de cada opció, dels dos algorismes, computem per a cada algorisme diferents temps d'execució variant la càrrega de treball, és a dir, el nombre d'expedients que processem; utilitzem la funció de *Pandas* "sample" per fer variar aquest nombre. D'aquesta manera, obtindrem unes corbes que ens permetran visualitzar la diferència en eficiència i deduir de forma aproximada el que en ciències de la computació es coneix com l'ordre de complexitat d'un algorisme. Obtenim els següents temps (utilitzant un processador mòbil i5-7200U a 2,5 GHz i 4GB de RAM a 2400 MHz):

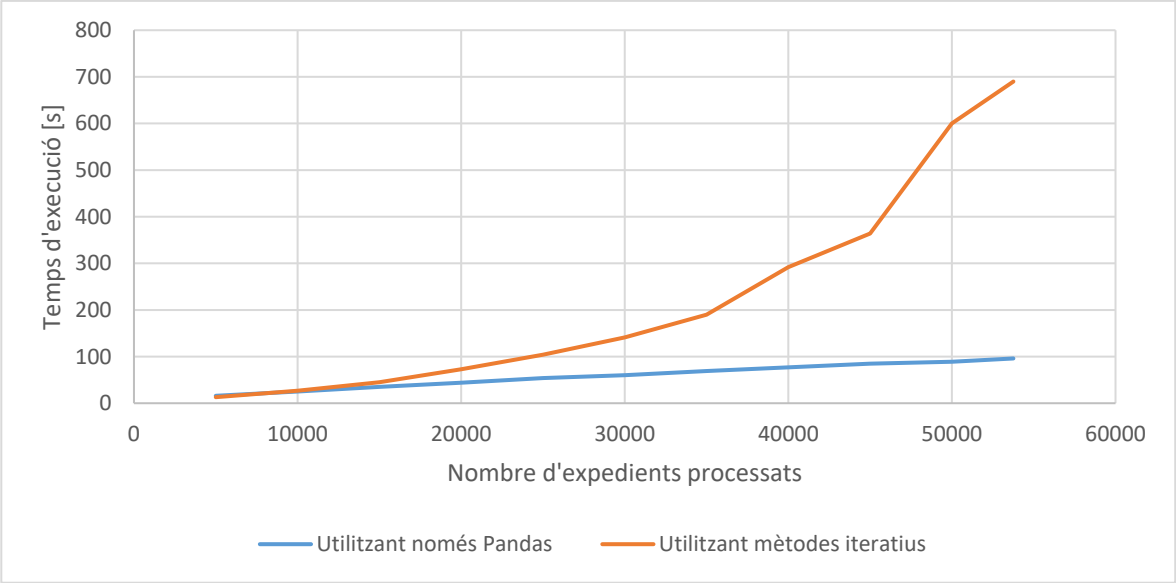


Figura 12: Temps d'execució de les dues opcions per al preprocessament de les dades.

Observem que l'ordre de complexitat de l'algorisme que només utilitza *Pandas* és lineal, *O(n)*, mentes que l'ordre de l'algorisme que fa ús de mètodes iteratius és polinòmic, *O(p(n))*. En altres paraules, si només utilitzem *Pandas* el temps d'execució augmenta linealment amb l'escala del *data set* que processem, mentre que si no utilitzem *Pandas* per a realitzar totes i cadascuna de les operacions sobre el *data set*, augmenta seguint una corba polinòmica. Aquests resultats emfatitzen doncs la importància d'utilitzar mètodes optimitzats en el tractament de dades sempre que sigui possible. Recordem que el codi que fa servir iteracions utilitza altres mètodes optimitzats de *Pandas* per a fer les mateixes operacions que fa "groupby", i que itera de manera eficient, eliminant en la mesura del possible iteracions innecessàries. Recordem també que el nostre *data set* és relativament petit; en conseqüència, encara que no siguem del tot eficients, podem processar els 53.758 expedients que tenim en uns 10 minuts; no obstant, donat que els mètodes iteratius no escalen linealment, si augmentéssim el nombre d'expedients el temps d'execució es faria ràpidament inviable.

3.6. *Data Frames* finals

Arribats a aquest punt ja tenim totes les dades que volem per construir el model: les notes obtingudes en la darrera convocatòria, aquestes mateixes notes expressades en percentatge respecte a la mitja, i el nombre de repeticions de cada assignatura. No obstant, ens falta estructurar aquestes dades de forma adequada, ens falta fer l'operació de *pivoting* que avançàvem al punt 3.4.

En efecte, en el nostre DF actual cada columna correspon a una convocatòria, i, per tal d'aplicar els algorismes de SVM, necessitem que cada fila correspongui a un estudiant, de

manera que les columnes continguin totes les dades relatives a un estudiant donat. Ho podem aconseguir de manera molt eficient i amb una única línia de codi gràcies al mètode de *Pandas* "pivoting" que hem explicat anteriorment. En concret, l'apliquem amb els següents paràmetres:

```
#Pivoting
FaseIniExp=FaseIni.pivot(index= 'CODI_EXPEDIENT', columns='CODI_UPC_UD', values=['NOTA_NUM_DEF', '%Mitja', 'Rep'])
```

És evident que "index" ha de ser igual a "CODI_EXPEDIENT", doncs volem que aquesta columna del DF, que identifica l'estudiant, sigui l'índex del nou DF. Els altres dos paràmetres de la funció, "columns" i "values", mereixen una breu explicació, doncs ens trobem en un cas lleugerament més complicat que el que hem explicat teòricament a l'apartat 3.4.

A diferencia del cas senzill que hem presentat anteriorment, en què només interessa una columna de dades, ens interessen, per a cada estudiant, les tres dades que acabem de mencionar: notes, notes en percentatge i nombre de repeticions. Aquestes tres dades són sempre en referència a un estudiant donat i a una assignatura donada. Així, fixem el paràmetre "columns" a "CODI_UPC_UD", per tal d'obtenir un DF en el què les files corresponen a estudiants i les columnes a assignatures, i el paràmetre "values" a una llista de les tres columnes que contenen les tres dades que volem. D'aquesta manera se'ns generarà un DF anomenat jeràrquic, en el què —per dir-ho ràpid— per una fila i columna determinades tenim tres valors, com si tinguéssim tres DFs amb idèntiques files i columnes superposats.

Els models SVM que utilitzarem, però, no admeten aquestes estructures jerarquitzades, així que hem de fer una darrera operació. Despleguem el DF jeràrquic en un DF normal augmentant el nombre de columnes que tenim (10, que corresponen a les 10 assignatures de primer any) segons la següent operació:

$$10\ columnes * 3\frac{dades}{columna} = \ 30\ columnes\ amb\ 1\frac{dada}{columna}$$

Seguint l'analogia gràfica, és com si poséssim els tres DFs superposats de costat. Ja tenim d'aquesta manera les dades en un format apte per a ser processat pels algorismes SVM.

Ens falten però encara les dades *Y*. Fins ara hem comentat el processament de les dades contingudes en el fitxer excel "qfaseini", és a dir, les dades de les assignatures de primer any, que són les nostres dades *X*. No obstant, hem de processar també el fitxer "qfasenoini", que conté les dades de les assignatures del tercer quadrimestre, que són les nostres dades *Y*. Es tracta però d'un procés més senzill.

Volem que les assignatures del tercer quadrimestre aprovades en primera convocatòria tinguin un valor de 0 i la resta —les suspeses en primera convocatòria— un valor de 1. Encara que pugui semblar contrari al sentit comú designar amb un 0 els aprovats i amb un 1 els

suspesos, ens facilitarà enormement el treball de modelatge i validació, doncs els casos de suspès són els nostres "casos positius", els casos que volem detectar, i les mètriques dels models es calculen sovint en referència a aquests casos.

Així doncs, trobem el nombre de convocatòries amb el mateix procediment que en el processament de les assignatures de primer any i assignem un 1 (valor suspès) als casos en què hi hagi més d'una convocatòria (fet que implica que la primera convocatòria es va suspendre). Assignem també un 1 (valor suspès) als casos en què la nota sigui inferior a 5 (això és necessari ja que pot ser que un estudiant suspengui assignatures del tercer quadrimestre i abandoni la carrera, no presentant-se a una segona convocatòria). D'altra banda, assignem un 0 (valor aprovat) als casos en que la nota es igual o superior a 5. Finalment, fem una operació de *pivoting* per obtenir el DF de les dades *Y* desitjat, en el que les files corresponen als estudiants, les columnes a les assignatures, i els valors són 0 o 1 en funció de si la primera convocatòria s'ha aprovat o suspès.

CODI_EXPEDIENT	240031	240032	240033	240131	240132	240133
226410	0	0	0	0	0	0
226431	0	0	0	0	0	1
226455	0	0	0	0	0	0
226464	0	0	0	1	0	0
226467	1	1	1	1	1	1
226472	0	0	0	0	0	0
226494	0	0	0	0	0	1
226495	0	0	0	1	0	1
226499	0	0	0	0	0	0
226515	0	0	0	0	0	0
226543	0	0	0	0	0	1
226635	0	0	0	0	0	0
226648	0	1	1	0	0	1

Taula 4: DF de dades Y.

En darrer lloc, utilitzem la funció de *Pandas* "merge" per unir els DFs de dades *X* i *Y*. No és necessari que els dos conjunts de dades es trobin en una mateixa variable; sí és necessari però que els índexs dels dos conjunts siguin el mateix (ja que per l'algorisme de modelització la fila Y_i del conjunt de dades *Y* és el resultat que s'ha d'obtenir per a la fila X_i del conjunt de dades *X*), i la funció "merge" és una de les maneres més fàcils i eficients d'aconseguir-ho.

4. Modelatge i validació

4.1. Obtenció dels models

Com hem argumentat a l'apartat 2.3.2, recorrerem al mètode de *k-fold cross validation* per aquesta darrera fase del projecte. Recordem que consisteix en separar les dades en *k* subconjunts i iterar per tots ells prenent cada cop un dels *k* subconjunts com *test set* i la resta com *training set.* La llibreria *Scikit-Learn* ofereix diferents alternatives per implementar aquest mètode de validació, que comentarem tot seguit.

Recordem però abans que l'objectiu d'aquest procés és entrenar un model determinat amb unes dades *training X* i *training Y*, i predir amb aquest model entrenat uns valors *Y* prenent com a entrada les dades *test X*, que es comparen amb els valors reals *test Y* per tal d'obtenir finalment la matriu de confusions, de la qual es deriven una sèrie de mètriques que ens donen informació sobre la qualitat d'aquest model determinat. Per tant, és un procés que haurem de repetir per a cada model candidat que prenguem en consideració; de fet, el sentit del procés rau en comparar les mètriques que obtenim per a un model amb les mètriques que obtenim per a un altre model, per poder així determinar quin és més adequat.

Scikit-Learn posseeix instruments que prenen en consideració aquest caràcter intrínsecament repetitiu dels mètodes de validació, el fet que s'apliquen a un model determinat però per a obtenir unes mètriques que es volen comparar amb les obtingudes per a altres models. En concret, la llibreria té el mètode "GridSearchCV", que permet automatitzar l'aplicació de la *k-fold cross validation* (o de qualsevol altre mètode de validació) a models entrenats amb paràmetres diferents. En el nostre cas, però, no només prendrem en consideració un tipus de model amb diferents paràmetres sinó que, com hem comentat a l'apartat 2.2.6, construirem diferents models amb diferents *kernels*. Podríem utilitzar la funció "GridSearchCV" pels models de cada *kernel*, però preferim en canvi utilitzar iterativament la funció "cross_validate" per a obtenir les mètriques de cada model concret. Les dues solucions són equivalents. Una última alternativa seria utilitzar la funció "cross_val_score"; seria una opció equivalent en termes de resultats, però no en termes de costos de computació, doncs aquesta última funció només ens dona el valor d'una mètrica d'un model concret (i requereix però fer tot el procés de validació per obtenir la matriu de confusions, que hauríem de repetir redundantment per cada mètrica).

"Cross_validate" ens permet doncs, amb una sola línia de codi, obtenir les mètriques ("score") d'un model *classifier* determinat ("Clf") amb el mètode de validació *k-fold* especificat ("kfold").

```
Clf=svm.SVC(kernel=k, C=c)
seed=5
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
scores=['accuracy', 'precision', 'recall', 'f1']
cv_results = model_selection.cross_validate(Clf, X, Y, cv=kfold, scoring=scores)
```

Els paràmetre “random_state” i “shuffle” de la funció “KFold” (funció que és la que especifica el mètode de validació que utilitzarà la funció “cross_validate”) serveixen per a forçar que la divisió de dades en els *k* subconjunts no sigui aleatòria, que sigui exactament la mateixa cada cop que cridem la funció amb el mateix paràmetre “random_state”, de manera que tots els models seran validats amb els mateixos subconjunts de dades, en igualtat de condicions. S’observarà que el valor de *k* escollit és 10. El valor de *k* per defecte és 5; el fixem a 10 per tal d’obtenir resultats més consistents, doncs el nostre *data set* no és molt gran.

S’observarà també que per a establir el model a validar utilitzem el objecte “svm.SVC”. Aquesta és l’objecte de *Scikit-Learn* que conté els algorismes *classifiers* SVM (ja hem comentat a l’apartat 2.2.1 que existeixen algorismes SVM per altres tipus de problemes que no són de classificació). Notem que fixem dos paràmetres del objecte, “kernel” i “C”.

En efecte, tal com avançàvem a l’apartat 2.2.6, construirem models diferents utilitzant els quatre *kernels* predeterminat que ofereix *Scikit-Learn*. D’altra banda, com hem explicat a l’apartat 2.2.5, el paràmetre *C* és el que fixa l’amplada del *soft margin*. Recordem que per valors de *C* petits, l’algorisme admetrà que molts punts quedin a la banda incorrecta del hiperplà; per valors de *C* elevats, el *soft margin* es redueix fins a ser inexistent i passem a tenir un algorisme *hard margin*, que intenta que tots els punts quedin a la banda correcta del hiperplà. Per cadascun dels quatre tipus de *kernel* amb els quals construirem models, doncs, haurem de provar tots els models concrets possibles que s’obtenen fent variar el valor de *C*. És per això que paràmetres del objecte “svm.SVC” es fixen mitjançant les variables “k” i “c”, que fem variar. Com comentàvem més a dalt, al utilitzar “cross_validate” i no “GridSearchCV”, hem d’iterar per tots els models que volem comparar per obtenir les mètriques de cadascun. És el que fem amb el codi següent, dintre del qual s’insereix el codi citat que conté la funció “cross_validate”:

```
for k in ['linear', 'poly', 'rbf', 'sigmoid']:
    c=0.0001
    while c<100000:

        #[...]

        c=c*2
```

S’observarà que fem variar els valors de *C* des de $C = 10^{-4}$ fins a valors d’ordre 10^4 amb una progressió geomètrica de raó 2. Es considera que aquest és un bon criteri per tal de trobar el valor òptim del paràmetre [6]. És a dir, per no deixar-nos models que obtindrien millors mètriques (recordem que cada valor de *C* correspon a un model concret).

Podem comprovar la validesa d'aquest criteri. Si fem variar *C* de manera més fina, provant més valors per tal de trobar el model òptim precís, observem que efectivament la diferència en les mètriques és negligible.

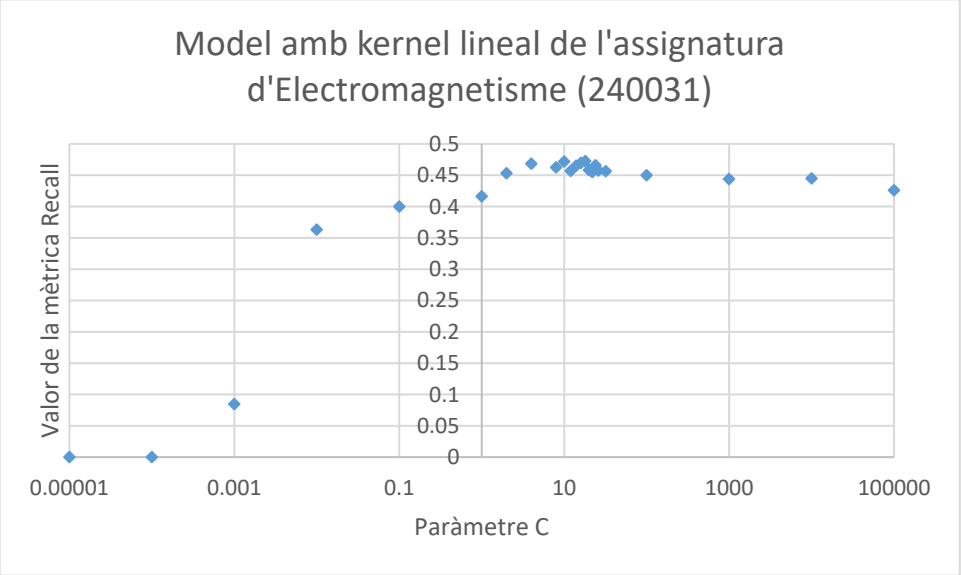


Figura 13: Gràfic dels valors de la mètrica recall *per a models de* kernel *lineal amb diferents valors del paràmetre C. L'assignatura modelitzada és Electromagnetisme. El model òptim es troba en* $C \approx 10$.

D'aquesta manera, utilitzant a l'interior de dos iteracions la funció "cross_validate", obtenim els valors de les quatre mètriques que ens interessen (*accuracy, precision, recall,* F_1) pels models SVM construïts amb quatre *kernels* diferents (lineal, polinòmic, "rbf" i sigmoide) i amb valors diferents del paràmetre *C.* Aquests models els hem de construir per cadascuna de les sis assignatures del Q3 que volem modelitzar. Les dues iteracions que hem comentat s'integren doncs en una tercera, que repeteix el procés per cada assignatura.

```
for a in ['240031', '240032', '240033', '240131', '240132', '240133']:
    for k in ['linear', 'poly', 'rbf', 'sigmoid']:
        Y=np.array(Dades[a])

        c=0.0001
        while c<100000:

            #[...]

            i=i*2
```

A banda del paràmetre *C*, existeixen alguns altres paràmetres secundaris en els models SVM. En particular, quan fem ús del *kernel* "rbf" apareix el paràmetre γ (veure fórmula a 2.2.6), que és inversament proporcional al radi d'influència de cada *support vector*. És a dir, per valors de γ petits els *supports vectors* no nuls de cada punt tindran una influència sobre la forma del hiperplà en punts llunyans, mentre que per valors de γ grans no en tindran cap. Per valors de

γ grans es produeix doncs *overfitting*, ja que el hiperplà s'ajusta més a cada punt. Podríem introduir en el nostre codi una quarta iteració per tenir en compte l'impacte d'aquest paràmetre, però no és necessari ja que, en l'objecte "svm.SVC" que és el nostre model, γ prendrà per defecte valors adequats en funció dels valor de *C* que fixem. Podem comprovar-ho amb un exemple.

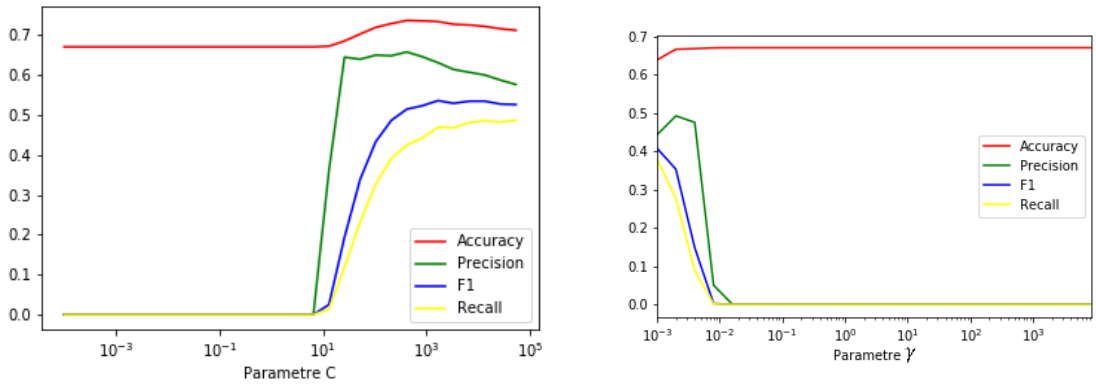


Figura 14: Valors de les mètriques utilitzant un kernel *"rbf" iterant C amb γ fixada automàticament (esquerra) i iterant γ amb C fixada (dreta)*.

En el cas de l'assignatura d'Electromagnetisme, quan construïm models amb *kernel* "rbf", obtenim que el millor model serà aquell amb $C \approx 10^3$. Si iterem el paràmetre γ amb *C* fixat a $C = 10^3$, observem que l'objecte "svm.SVC" ja ens havia escollit la γ optima.

També tenim un paràmetre γ en el *kernel* sigmoide (veure fórmula a 2.2.6), però l'objecte "svm.SVC" el fixa igualment per defecte a un valor adequat.

D'altra banda, quan fem ús del *kernel* polinòmic apareix el paràmetre n, que correspon al grau del polinomi (veure fórmula a 2.2.6). Aquest paràmetre es fixa per defecte a $n = 3$. No provarem quins models obtindríem fent variar aquest paràmetre ja que, com es veurà, en la majoria dels casos els models obtinguts amb *kernel* polinòmic seran inferiors als obtinguts amb altres *kernels*; a més els models amb *kernel* polinòmic seran també els models amb costos de computació més elevats (amb temps d'execució al voltant de tres hores per la iteració de *C*), i elevar el paràmetre n augmentaria aquests costos notablement.

Per acabar amb la qüestió dels paràmetres secundaris, fem notar que en els *kernels* polinòmic i sigmoide apareix també un coeficient c_0, que no té però un efecte significatiu en el model.

Finalment, construirem un segon grup de models repetint tot el procés que acabem d'explicar amb unes variables d'entrada diferents. Encara que els algorismes SVM funcionin bé amb un nombre gran de variables *X* (la relació entre la complexitat del model i el seu nombre de variables és una relació d'ordre lineal) i, per tant, el fet d'utilitzar 30 variables no és cap inconvenient, construirem també models alternatius amb 10 variables. Disposarem així de

més models alhora de seleccionar l'òptim, que en alguns casos serà un model de 10 variables; d'altra banda, el fet mateix de disposar de més models, encara que no els seleccionem per sobre dels de 30 variables, és positiu, doncs disposem així d'alternatives amb comportaments diferents.

Recordem quines són les nostres 30 variables: tenim les notes en les 10 assignatures de primer any i, per cada assignatura, la nota expressada en percentatge respecte a la mitja de la convocatòria i el nombre de convocatòries a les quals s'ha presentat l'estudiant. Podem reduir aquestes 30 variables a 10 d'acord amb la següent fórmula:

$$X_i = Nota_i * \%Mitja_i - \#Convocatories_i^2 \quad \text{(eq. 24)}$$

Com acabem de comentar, seguirem exactament el mateix procediment d'utilitzar quatre *kernels* i iterar els valors de *C* per obtenir els models de 10 variables.

Cal destacar que els costos de computació del model de 10 variables no van ser inferiors als costos del de 30; al contrari, van ser notablement superiors i de fet vam haver d'implementar un mecanisme de *timeout* per sortir de les iteracions quan la funció "cross_validate" trigui dues hores a executar-se. Al reduir el nombre de variables, estem reduint la informació del model, i en la majoria dels casos aquesta reducció d'informació pesa més que el benefici de treballar en espais vectorials de dimensió inferior (que és el que estem fent al reduir el nombre de variables). En uns quants casos particulars, però, el model de 10 variables obté resultats pràcticament idèntics i fins i tot millors al model de 30 amb temps d'execució menors.

D'altra banda és més fàcil donar un sentit als seus resultats, doncs cada variable X_i (que determinarà al final les prediccions a les què volem arribar) dependrà únicament dels resultats acadèmic obtinguts en una sola assignatura de primer curs. Per exemple, si el model prediu un suspens per a un cert expedient, en el model de 10 variables podem fàcilment jugar amb els valors de les variables per analitzar en quines assignatures l'estudiant hauria d'haver obtingut millors resultats per a que el model predigués un aprovat i no un suspens, i fins i tot hipotèticament aconsellar aleshores en quines àrees ha de millorar.

Tots els resultats es recullen en l'annex 1, en forma de taules resum per a cada assignatura. Reproduirem però al cos del text els resultats que considerem més oportuns, com ja hem fet en aquest apartat.

Un cop obtinguts tots els models, procedim a comparar les seves mètriques per a seleccionar el millor model per a cada assignatura. Expliquem a continuació quins seran els criteris que utilitzarem en aquesta comparació i per què hem decidit centrar-nos en les quatre mètriques escollides.

4.2. Criteris d'avaluació

És important recordar en aquest apartat que prenem com a casos positius els suspensos, doncs així ens resulta més fàcil i intuïtiu avaluar els models amb les mètriques.

Per seleccionar els models òptims prioritzarem els valors obtinguts per cada model en la mètrica *recall*, tot comprovant que els valors alts de *recall* no s'aconsegueixen sacrificant els valors de la mètrica *precision*; en aquest sentit la mètrica F_1, que recordem és la mitja geomètrica d'aquestes dues mètriques, ens ajudarà a analitzar el *trade-off* que paguem per tenir bons valors de *recall*. Addicionalment, la mètrica *accuracy* ens informarà de la qualitat general del model.

Per què seguirem aquest criteri? Perquè l'objectiu principal del nostre projecte és construir models que siguin capaços d'alertar de possibles suspensos en assignatures del tercer quadrimestre, que hem codificat com el nostre cas positiu. Per tant prioritzarem que el nostre model no deixi de preveure cap suspens real, que és el que indica la mètrica *recall* (veure apartat 2.3.3). Al mateix temps, però, no volem que el nostre model predigui un nombre massa elevat de suspensos, molts d'ells incorrectes, doncs aleshores no seria de gran utilitat; no volem que el nostre model predigui molts suspensos (casos positius) que no seran tals, i això és el que indica la mètrica *precision* (veure apartat 2.3.3).

En resum, escollir models basant-nos d'aquesta manera en les mètriques de *recall* i *precision* és trobar el punt adequat entre no deixar de preveure cap suspens real i proporcionar informació útil, no preveient un suspens per tot estudiant que no tingui un rendiment acadèmic alt.

Ho podem acabar d'il·lustrar amb un exemple concret. Per a l'assignatura de Mecànica, obtenim que el millor model de tipus "linear" —amb *recall* més alt i valor de *precision* és el de paràmetre $C = 0.0008$, al qual correspon la següent matriu de confusions:

$$M = \begin{pmatrix} 680 & 563 \\ 246 & 979 \end{pmatrix}$$

D'aquesta matriu de confusions es deriven els valors de les mètriques, que seran els que analitzarem. Segons les formules de l'apartat 2.3.3:

$$Recall = \frac{979}{979 + 246} = 0.7992$$

$$Precision = \frac{979}{979 + 563} = 0.6393$$

Un valor de *recall* de $79{,}92\%$ ens diu que el model detecta el $79{,}92\%$ dels casos reals de

suspens, del total que hauria d'haver detectat, 979 de 1225 en aquest cas concret.

Un valor de *precision* de $63{,}93\%$ ens diu que un $63{,}93\%$ dels casos que el model prediu com a suspensos, són efectivament —acaben sent— casos de suspens real; en aquest cas concret el model prediu 1542 suspensos, dels quals 979 ho acaben sent realment.

Idealment, voldríem és clar que el model predigués únicament els 979 suspensos. Donat que això és impossible, ens fixem com a prioritat que el model predigui la majoria d'aquests casos de suspens —i en aquest cas concret podem aconseguir que predigui 4 de cada 5—, encara que això provoqui falsos positius, casos en que el model alertarà d'un suspens en principi equivocadament, 563 en aquest cas, o el que és el mateix, aproximadament 1 de cada 3 suspensos que prediu, com ens indica la mètrica *precision*.

4.3. Consideracions generals sobre els models obtinguts

Abans de procedir a les comparacions entre els diferents models per cada assignatura, farem a continuació alguns comentaris de caràcter general sobre el comportament dels models.

4.3.1. Costos computacionals i efecte del paràmetre *C*

Pel que fa als costos de computació, els models amb *kernels* "rbf" i sigmoide tenen temps d'execució molt baixos, de l'ordre de segons; els models amb *kernel* lineal tenen un temps més elevat, de l'ordre de minuts (o de hores en el cas dels models de 10 variables); i finalment els models amb *kernel* polinòmic tenen els temps d'execució més elevat, de l'ordre d'hores.

En general, els resultats milloren a mesura que *C* augmenta i reduïm per tant l'amplada dels *soft margins*. És raonable que sigui així, doncs al reduir els *soft margins* fem que els hiperplans separin de manera més estricta les dades d'entrenament, i estan aleshores més ben definits que per valors de *C* petits. Al augmentar *C* incorrem però en el risc de *overfitting*, doncs precisament estem fent que el model separi de manera més rigorosa les dades d'entrenament, augmentant per tant la seva complexitat i fent que s'adapti més al *training set*. Utilitzar *k-fold cross validation* en comptes de *hold out validation* ajuda a evitar aquest problema, però en general no escollirem models amb valors de *C* elevats si tenim l'alternativa d'escollir models amb mètriques similars però un paràmetre *C* més baix. A banda del risc de *overfitting*, els models amb paràmetres *C* més petits tindran costos de computació notablement inferiors, doncs la complexitat del model serà inferior.

4.3.2. Pics en el valor de *precision*

Observem que pràcticament tots els models presenten un pic sobtat en el valor de la mètrica *precision* al augmentar el valor de *C*, mentre les altres mètriques es mantenen properes a 0.

No és d'estranyar: quan el model es faci prou complex com per començar a predir casos de suspens, els pocs casos que predigui seran sempre suspensos reals.

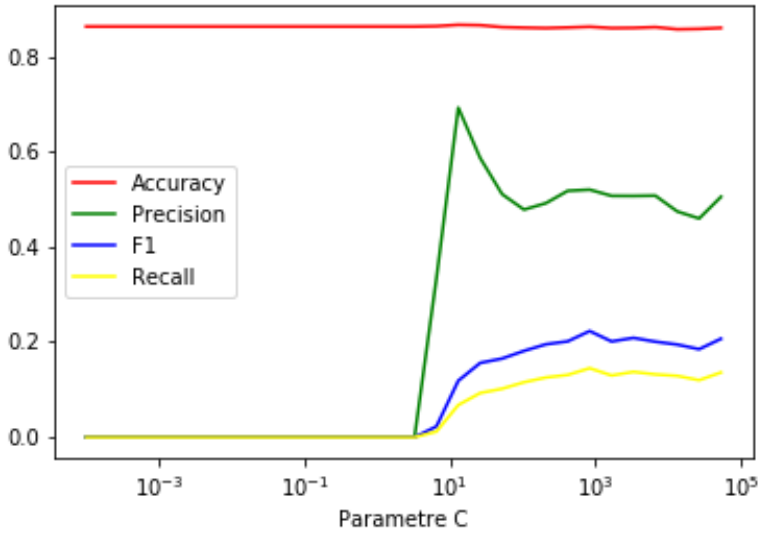


Figura 15: Valors de les mètriques utilitzant un kernel *lineal en l'assignatura de Mètodes numèrics.*

Per exemple, el models amb *kernel* lineal per a $C = 13{,}1072$ té un elevat valor de *precision*.

Si calculem la matriu de confusions per aquest valor de *C*, obtenim:

$$M = \begin{pmatrix} 2115 & 14 \\ 316 & 23 \end{pmatrix}$$

La mètrica *precision* d'aquesta matriu val $62{,}16\%$. Això ens diu simplement que dels pocs suspensos que el model prediu, 37, la majoria d'ells, 23, són suspensos reals.

4.3.3. Models amb *kernel* polinòmic

Els models amb *kernel* polinòmic són marcadament millors a mesura que augmentem *C*. En molts casos, semblaria que amb una *C* més elevada podrien arribar a superar models amb altres *kernels*.

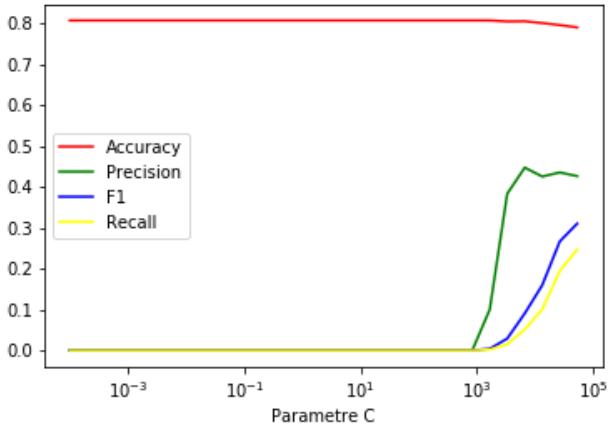


Figura 16: Models amb kernel *polinòmic per l'assignatura d'Equacions diferencials.*

No obstant, a banda dels problemes de *overfitting* i de costos computacionals que ja hem comentat, pot ser simplement que al augmentar C més enllà de 10^5 sigui senzillament impossible trobar un hiperplà que separi adequadament les dades. A l'apartat 2.2.3 hem explicat que els algorismes SVM *soft margin* van aparèixer històricament com una generalització dels de *hard margin* (que és el que obtenim al fer tendir C a l'infinit) degut a que, en la gran majoria de problemes, és impossible separar absolutament totes les dades d'un tipus de les de l'altre. El nostre cas no és cap excepció. Vam provar valors de C de l'ordre de 10^5 i després de 24 hores d'execució no s'havia pogut entrenar el model.

4.3.4. Models amb *kernel* sigmoide

Els models amb *kernel* sigmoide presenten en general un comportament brusc, amb uns mateixos valors de les mètriques a partir d'una certa C.

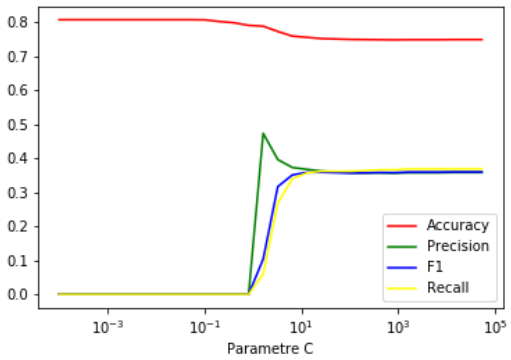


Figura 17: Models amb kernel *sigmoide per l'assignatura d'Equacions diferencials.*

Aquest comportament es deu a que la funció sigmoide, que hem escrit expressada amb la tangent hiperbòlica (veure fórmula a 2.2.6), és de fet la mateixa funció que utilitza la regressió logística, que com dèiem a l'apartat 2.2.2 dona sempre com a resultat valors entre 0 i 1. De

fet, i més concretament, dona sempre resultats molt propers a 0 i 1.

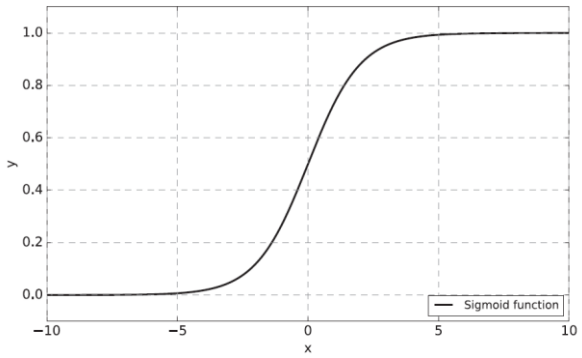


Figura 18: Gràfica d'una funció sigmoide estàndard. Extret de [14].

En conseqüència, si el *kernel*, la funció que substitueix el producte escalar en l'algorisme SVM, té aquest comportament, al arribar a un cert valor de *C* passarem de tenir un *soft margin* molt ample a tenir un *hard margin*. Podríem evitar-ho normalitzant tots els valors *X* i fent així que es trobin en la zona central de la funció sigmoide, en la qual no val 0 o 1. Al dur a terme aquesta operació, però, els models que se'n deriven no prediuen pràcticament cap cas positiu, fins i tot per valors elevats de *C*. Únicament en l'assignatura de Mecànica —que té un nombre de suspensos molt més alt que la resta— obteníem models que predeien casos positius.

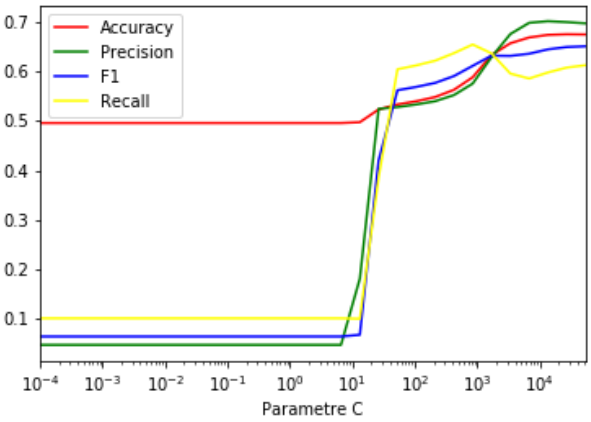


Figura 19: Models amb kernel *sigmoide i valors X normalitzats per l'assignatura de Mecànica.*

En conseqüència, no hem fet aquesta operació de normalitzar les dades *X* i hem mantingut el comportament brusc dels models sigmoides. En alguns casos, són de fet els models que donen millors resultats. Recordem d'altra banda que un model *hard margin* no és intrínsecament pitjor que un model *soft margin*, per bé que sí més propens al *overfitting*.

4.3.5. La relació entre els models de 30 i de 10 variables

En alguns casos, el gràfic dels valors de les mètriques en funció de *C* dels models de 10 variables sembla un desplaçament cap a l'esquerra del gràfic dels models de 30 variables.

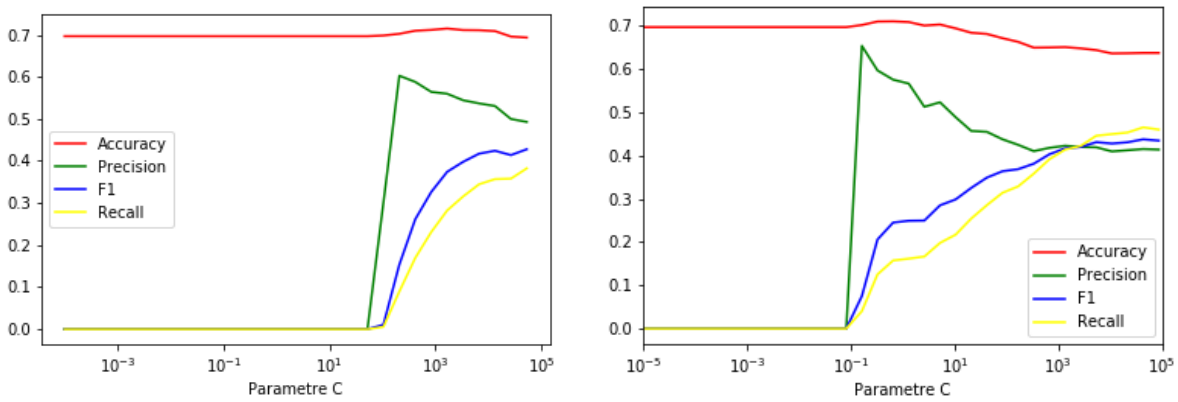


Figura 20: Models amb kernel *"rbf" de 30 (esquerra) i 10 (dreta) variables de l'assignatura Materials.*

Al passar de 30 a 10 variables estem imposant relacions entre variables i doncs perdent informació, provocant *underfitting* (fenomen invers al de *overfitting*). En conseqüència, el model no ha d'arribar al grau de complexitat al que necessita arribar en el cas de 30 variables per començar a preveure suspensos: per un mateix valor de *C* (paràmetre que és directament proporcional a la complexitat), per exemple $C = 10^1$ en la Figura 20, el model de 10 variables prediu suspensos i doncs té un *recall* no nul, mentre que el model de 30 variables encara no ha trobat una manera separar els punts en l'espai de 30 dimensions que produeixi una regió de suspensos prou important. Un cop el model de 30 variables és prou complex per generar aquesta regió, però, començarà a preveure suspensos de manera més eficient que el model de 10 variables: observem en efecte que el valor de *recall*, a partir de $C \approx 10^2$, puja més ràpidament que no pas en el model de 10 variables a partir de $C \approx 10^{-1}$.

En alguns casos, com en l'exemple escollit de la Figura 20, encara que la mètrica *recall* pugi més lentament, amb menys pendent, en els models de 10 variables, el fet de començar a pujar abans fa que pugui arribar a valors més alts. En altres paraules, encara que els models de 10 variables amb paràmetres *C* baixos presentin *underfitting* i tinguin un valor de *recall* que no és nul però que és molt baix, al augmentar *C* podem arribar a valors de *recall* superiors als dels models corresponents de 30 variables.

D'altra banda, com hem explicat anteriorment, els models de 10 variables també tenen l'avantatge de ser més fàcils d'interpretar pel que fa al funcionament intern del model. Com veurem a continuació, pot ser complicat interpretar models que treballen amb tantes dimensions.

4.3.6. Formes del hiperplà pels diferents *kernels*

A l'apartat 2.2.6 tenim un gràfic dels hiperplans generats per cada *kernel* en un problema 2D, amb dues variables d'entrada. El nostre és un problema de 30 dimensions, amb 30 variables,

i per tant els hiperplans no es poden visualitzar ni sobre el paper ni mentalment. Els principis, però, són els mateixos: en el *kernel* lineal, és el producte escalar el que defineix el hiperplà (com hem vist a l'explicació teòrica dels SVM, veure 2.2.4), que tindrà doncs propietats lineals; si usem un *kernel* polinòmic, el hiperplà estarà definit per un polinomi, i doncs presentarà corbes; si usem un *kernel* "rbf", el hiperplà estarà definit per una funció radial gaussiana, i doncs tindrà formes circulars; si usem un *kernel* sigmoide, el hiperplà estarà definit per la funció sigmoide, i presentarà formes quasi-lineals, però amb canvis en les rectes.

Podem visualitzar de manera simplificada el hiperplà representant-lo en dues dimensions. En el següent exemple considerem l'assignatura de Mecànica i, únicament, dues variables X, les notes en les assignatures de primer any de Mecànica fonamental i de Termodinàmica fonamental. Creem una graella que va des de 5 a 10, els possibles valors de les dues notes de primer any, i, amb els millors models de cada *kernel*, prediem tots els valors aprovat/suspès d'aquesta graella. Ens falten evidentment 28 variables per poder fer les prediccions. Utilitzem un mètode molt senzill per obtenir-los de manera artificial: busquem en les dades de què disposem alumnes que tinguin notes de Mecànica fonamental i de Termodinàmica fonamental similars a les que estem utilitzant per la predicció, i prenem aleshores la mitja de les variables *X* d'aquests alumnes per les 28 variables que ens falten. Obtenim així les regions d'aprovat i de suspès i doncs una aproximació del hiperplà.

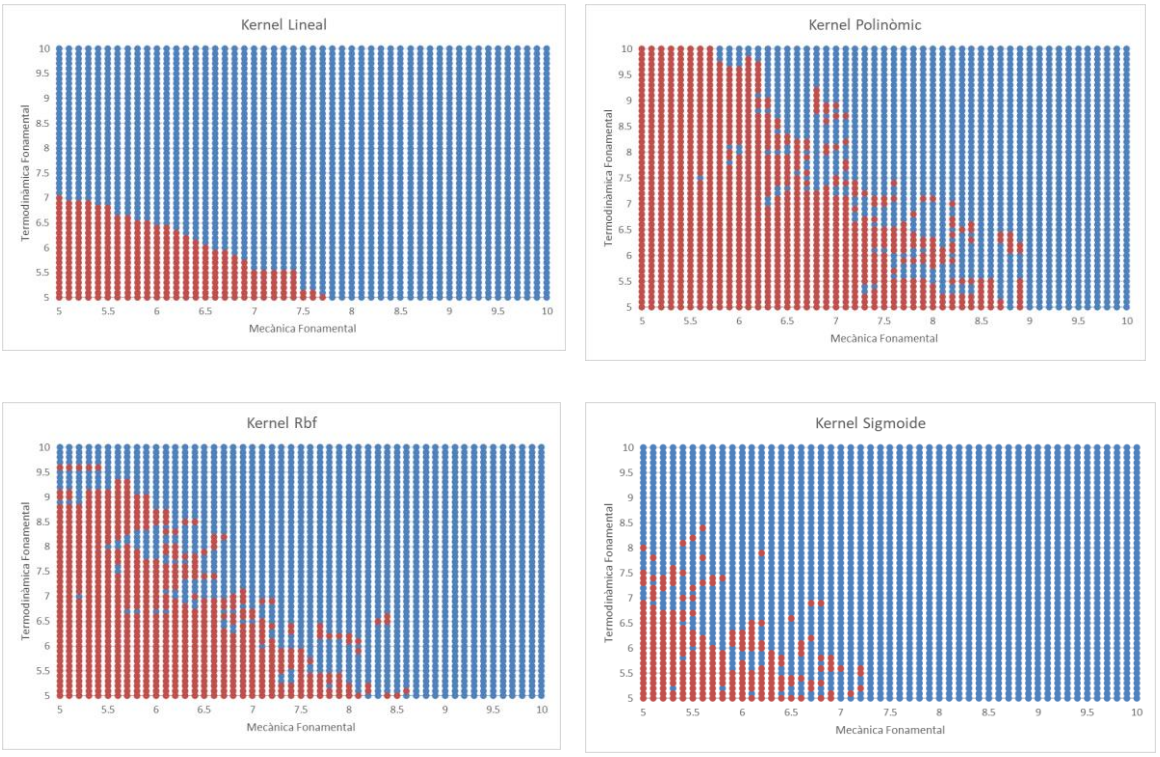


Figura 21: Formes simplificades dels hiperplans pels diferents kernels.

Observem que podem reconèixer en les regions de cada *kernel*, particularment en les seves fronteres, detalls similars les formes que a l'exemple 2D de l'apartat 2.2.6 caracteritzen tota la regió. Per exemple, en el punt (6, 9.5) del *kernel* polinòmic observem clarament una corba polinòmica; en el punt (5.5, 8.5) del *kernel* "rbf" observem un sortint amb una forma radial; i en el punt (5.5, 7) del *kernel* sigmoide observem uns quants punts de tipus suspens que formen una diagonal creixent. Pel que fa al *kernel* lineal, encara que la forma en conjunt de la regió és més similar a la del exemple 2D, observem en el punt (7, 5.5) que la recta diagonal principal es veu interrompuda per una recta horitzontal. Tots aquests fenòmens es deuen és clar a la interferència de les altres 28 dimensions que no estan representades en els gràfics.

En resum, quan treballem amb més dimensions els hiperplans de cada *kernel* continuen tenint per suposat les mateixes propietats que podem observar en l'espai 2D, però de manera més complexa, generant més rectes lineals, més corbes polinòmiques, més radis i més agrupaments intersecants. No podem dir de cap d'aquests hiperplans que sigui en principi millor que un altre, haurem d'obtenir les mètriques per tots ells i comparar analíticament els resultats.

4.4. Selecció dels models òptims

4.4.1. Electromagnetisme (240031)

Tenim moltes opcions per a l'assignatura d'Electromagnetisme, doncs podem assolir amb molts models diferents valors de *recall* similars, propers al 50%. El model que arriba a un valor més alt de *recall* és el model de 10 variables, *kernel* sigmoide i $C = 0.3277$, que assoleix un *recall* del 52.99%. Podem escollir-lo com l'òptim, doncs té un valor de *precision* acceptable, del 54.43%. El model escollit, aleshores, preveurà més de la meitat dels suspensos reals i s'equivocarà en menys de la meitat dels casos.

Com hem argumentat, prioritzem el valor de *recall*, detectar tants casos de suspensos reals com puguem. Tenim alternatives amb millors valors de *precision*, però els punts percentuals que guanyem en *recall* amb el model escollit respecte a les alternatives justifiquen l'elecció. La diferència en *precision* no és prou important per no escollir el model amb *recall* més alt. Les alternatives són els models de 30 variables i *kernels* lineal, polinòmic i "rbf"; es poden veure els valors exactes a la Taula 5.

Kernel	*C*	*Precision*	*Recall*
Lineal	6,5536	64,18%	47,01%
Polinòmic	53 687	57,97%	50,22%
“Rbf”	13 421	59,95%	48,57%

Taula 5: Models alternatius de 30 variables.

Una altra raó per no escollir aquests altres models és, en el cas dels dos últims, el alt valor del paràmetre *C*, que pot provocar problemes de *overfitting*, encara que amb el mètode de validació 10-*fold cross validation* probablement els hem evitat. D’altra banda, el cost computacional del model polinòmic és molt més elevat que el dels altres models.

4.4.2. Mètodes numèrics (240032)

Degut al relativament baix índex de suspensos en l’assignatura, 357 de 2551, un 14%, el més baix de les sis que estudiem, és difícil obtenir un model que predigui bona part d’aquests pocs suspensos reals.

Encara que els models sigmoide de 30 variables i “rbf” de 10 variables siguin els que ens proporcionen millors valors de *recall*, el model escollit és de 30 variables, *kernel* lineal i $C = 838.86$, ja que existeix una gran diferència en els valors de *precision*. Els valors de *precision* dels models amb *kernel* sigmoide i “rbf” no són acceptables, són inferiors al 25%; 3 de cada 4 prediccions de suspens serien incorrectes. Un altre indicador de què són models amb massa errors és el fet que la mètrica *accuracy* baixa per sota dels valors habituals.

Aquest és un estrany cas en què la regressió logística ens ofereix resultats que són en alguns aspectes millors que els dels SVM. En efecte, ens ofereix millor *precision* respecte al model escollit, però en aquest cas la diferència de no és prou significativa.

Nombre de variables	Algorisme	*C*	*Accuracy*	*Precision*	*Recall*
30	SVM lineal	838,86	86,22%	52,00%	14,51%
30	SVM sigmoide	52,42	79,05%	24,15%	23,09%
10	SVM “rbf”	85 899	75,89%	22,52%	30,08%
30	Regressió logística	13,11	86,63%	58,42%	12,43%

Taula 6: Model escollit (primera fila) i models considerats.

El model “rbf” de 10 variables segueix un comportament com el descrit a l’apartat 4.3.5, però, encara que tingui un valor elevat de *C*, no aconsegueix superar el *underfitting* que apareix en els models de 10 variables.

4.4.3. Materials (240033)

Així com en el cas anterior, en l’assignatura de Mètodes numèrics, hem hagut de descartar el model de 10 variables “rbf” perquè tot i donar el *recall* màxim presentava *underfitting*, en aquest cas és el model escollit ja que arriba a superar aquests problemes de *underfitting* i ofereix de nou el valor màxim de *recall*.

En concret, el model escollit és el de 10 variables, *kernel* “rbf”, i $C = 42\,949$, i ens permet obtenir un *recall* del 46.50% amb un valor baix però acceptable de *precision*, un 41.49%, valor que d’altra banda és comparable al que ens ofereixen els altres models. Podrem doncs preveure pràcticament la meitat dels suspensos reals.

Com és habitual, tenim alternatives que ens ofereixen millor *precision* sacrificant *recall*. Es tracta en aquest cas dels models de 30 variables lineal i sigmoide. Els models de 30 variables polinòmic i “rbf” ens ofereixen valors similars, però cap avantatge real (entenent que estem en el fons en un problema d’optimització, podem dir que són solucions dominades i no *trade-offs*). El model lineal ens ofereix una bona *precision*, permetent que més de la meitat de les nostres prediccions de suspens siguin correctes, i el model sigmoide ens ofereix un punt intermedi, amb un *recall* millor que el lineal però pitjor que el model escollit i una *precision* pitjor que el lineal però millor que el model escollit. De nou, però, davant de petites diferències prima el valor de *recall*.

Kernel	*C*	*Precision*	*Recall*
Lineal	838,86	55,48%	38,09%
Sigmoide	419,43	44,09%	44,50%

Taula 7: Models alternatius de 30 variables.

4.4.4. Equacions diferencials (240131)

En el cas de l'assignatura d'Equacions diferencials trobem uns models que sobresurten notablement per sobre de la resta, els models de 30 variables amb *kernel* lineal.

En concret, escollirem el model de 30 variables i *kernel* lineal amb $C = 53\,687$, que té un *recall* del 27.47% i una *precision* del 53.87%. Aconseguim doncs predir 1 de cada 4 suspensos reals i donar unes prediccions de suspens que són certes en més de la meitat del casos.

És cert que tenim models que ofereixen valors de *recall* superiors, els dos models "rbf" i el model de 30 variables sigmoide. Cap d'ells però té valors de *precision* acceptables. El millor de tots ells és clarament el sigmoide de 30 variables amb $C = 26.21$, que a banda d'un *recall* del 36.16% (els tres models citats tenen valors similars de *recall*) arriba a assolir una *precision* del 36.22%. No obstant, es tracta igualment d'un valor massa baix, només 1 de cada 3 suspensos previstos ho són realment. De nou, la mètrica *accuracy* (que torna a prendre valors per sota de l'habitual) ens serveix com a símptoma d'un model mal ajustat, amb *underfitting*, que encara que pot preveure un percentatge major dels suspensos, s'equivoca sovint.

4.4.5. Informàtica (240132)

El cas de l'assignatura de Informàtica és pràcticament calcat a l'anterior, l'assignatura d'Equacions diferencials: els model de 30 variables i *kernel* lineal ofereixen inequívocament els millors resultats, i els models "rbf" i sigmoide ofereixen valors de *recall* més alts, però estan mal ajustats; en aquest cas, només el model de 10 variables "rbf" ofereix un valor de *recall* més alt, en un altre exemple del que hem comentat a l'apartat 4.3.5.

En concret, escollim com a model òptim el model de 30 variables amb *kernel* lineal i paràmetre $C = 104.9$, que ens dona un *recall* del 26.03% i una *precision* del 51.76%.

Com en el cas de l'assignatura d'Equacions diferencials, entre els models mal ajustats que ofereixen millor *recall*, el millor és el sigmoide, amb un paràmetre *C* que és també similar, $C = 52.43$. Ens dona un *recall* del 35.88% i una *precision* també massa baixa, del 36.67%.

Si comparem les dades de les assignatures de Informàtica i d'Equacions diferencials,

comprovem que en efecte són molt similars. La primera té 536 suspensos i la segona 499, 200 dels quals són compartits.

4.4.6. Mecànica (240133)

Així com és difícil obtenir models amb mètriques altes en l'assignatura de Mètodes numèrics degut al baix índex de suspens, en l'assignatura de Mecànica tots els models ens donen mètriques molt superiors als de la resta d'assignatures. Mecànica és l'assignatura amb un índex de suspens més elevat 1275 de 2551, un 50%, i doncs és fàcil que els models detectin part d'aquest alt nombre de casos positius.

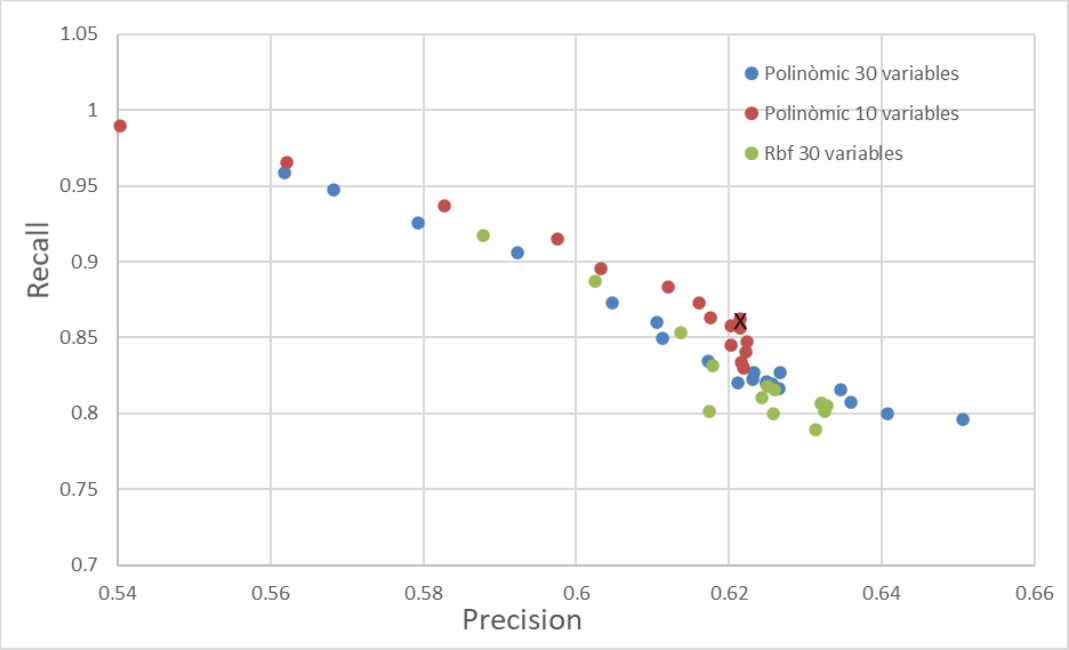


Figura 22: Valors de Precision *i* Recall *dels millors models per a l'assignatura de Mecànica. El model seleccionat s'indica amb una creu.*

Observem però que són tres els models que ens ofereixen els millors resultats: els dos models polinòmics i el model "rbf" de 30 variables. Donat que ens trobem en un cas en què trobar l'òptim és més complicat —ja que tenim molts models concrets que ens ofereixen resultats similars—, fem un gràfic dels valors més alts de *recall* i *precision* que obtenim amb aquests tres tipus de models.

Estem davant d'un problema prototípic d'optimització, i podem veure que la frontera de Pareto està constituïda pràcticament de manera exclusiva per punts del model polinòmic de 10 variables. Més específicament, podem dir que, fins a valors de *precision* de 0.62, la frontera de Pareto està en efecte constituïda exclusivament per punts del model polinòmic de 10

variables. Com que prioritzem el valor de *recall* al de *precision*, descartem els altres dos tipus de models i ens centrem en aquest, que ens oferirà els millors resultats sense *trade-offs* per a valors de *precision* inferiors a 0.62.

I quin model específic seleccionem aleshores? Si seguíssim cegament el criteri que hem utilitzat fins ara, escolliríem el model que ens dona el valor de *recall* més alt, proper a 1. Estem estudiant models que difereixen només 8 punts percentuals en *precision*, i davant de diferències petites prioritzem el *recall*. No obstant, aquest criteri l'utilitzàvem per discriminar entre models que ens oferien per exemple un 35% i un 20% de *recall*, però ara ens movem en valors del 90%. I els models del gràfic s'obtenen amb valors del paràmetre *C* molt petits. Podem veure que tenim uns quants punts del model polinòmic de 10 variables molt espaiats i després un agrupament. Aquest agrupament l'interpretem com els valors de *C* per als quals deixem de tenir *underfitting* (encara que no sigui tan sever com en altres casos), i doncs seleccionem el model que pertany a aquest agrupament amb mètriques més altes.

Aquest model és —és clar— un *kernel* polinòmic amb 10 variables, i amb paràmetre $C = 0.00256$, i ens dona en concret un *recall* del 86.26% i una *precision* del 62.15%.

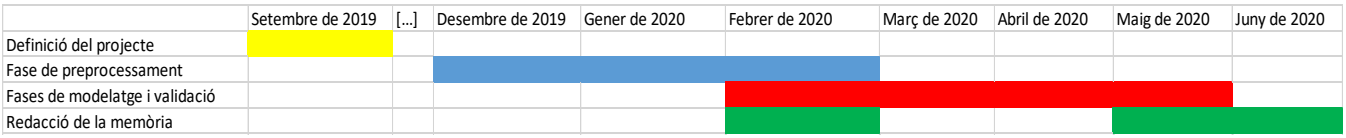
En el cas de l'assignatura de Mecànica aconseguim doncs preveure la gran majoria de suspensos i fer-ho amb una bona precisió, preveient correctament prop de 2 de cada 3 suspensos.

4.4.7. Taula resum dels models seleccionats

Assignatura	**Model seleccionat**	***Accuracy***	***Precision***	F_1	***Recall***
Electromagnetisme	10 variables, *kernel* sigmoide i C=0.3277	69.89%	54.43%	53.66%	52.99%
Mètodes numèrics	30 variables, *kernel* lineal i C=838.86	86.22%	52.00%	22.31%	14.51%
Materials	10 variables, *kernel* "rbf", i C=42949	63.69%	41.49%	43.74%	46.50%
Equacions diferencials	30 variables, *kernel* lineal i C=53687	81.52%	53.87%	35.95%	27.47%
Informàtica	30 variables, *kernel* lineal i C=104.86	79.78%	51.76%	34.29%	26.03%
Mecànica	10 variables, *kernel* polinòmic, i C=0.00256	67.06%	62.15%	72.19%	86.26%

5. Planificació i pressupost

En el següent diagrama de Gantt es pot observar la dedicació horària al projecte



Taula 8: Diagrama de Gantt de la planificación del projecte.

Els costos del present projecte són relativament baixos.

Podem fer una estimació del suposat cost de recursos humans. Prenem un preu d'hora de 30€ i estimem que el nombre d'hores totals és de 300. D'altra banda, tenim els costos materials. Estimem un cost en fungibles de 20€ i considerem el cost dels ordinadors que es requeririen. A la pràctica, hem usat un portàtil de segona mà de 400 euros i un ordinador de sobretaula de 300 euros. Estem estudiant però els costos suposats del projecte. Idealment, voldríem un únic ordinador amb una CPU amb un nombre relativament elevat de *cores*, doncs el software utilitzat els rendibilitzaria, per exemple una Ryzen 5 3600. Estimem doncs el cost de l'ordinador en uns 500 euros. Finalment, no tenim costos de llicències doncs no hem utilitzat software de pagament.

Recursos humans		
30€/h	300h	9000
Costos materials		
Ordinador		500
Fungibles		20
Total		9,520 €

Taula 9: Pressupost del projecte.

6. Impacte ambiental

L'impacte ambiental del projecte és mínim. A banda dels fungibles, que considerem que tenen un impacte ambiental menyspreable si els reciclem, el impacte ambiental del projecte es redueix al consum elèctric i a les emissions de gasos d'efecte hivernacle associades. Considerem que l'ordinador del projecte tindria un consum mig de 300W, i que el 90% de les hores de treball està encès. Utilitzem dades públiques del 2019 dels gasos d'efecte hivernacle generats en el consum elèctric: s'estimen en 241 grams CO_2/kWh. Hauríem generat aleshores 19.5 kg de CO_2.

Conclusions

Hem executat totes les fases previstes del projecte, i hem aconseguit per tant trobar models SVM òptims per a la predicció de casos de suspens a l'inici del segon any del Grau en Enginyeria en Tecnologies Industrials, que era el nostre objectiu inicial.

Són models òptims en tant que són els millors models per al nostre objectiu (predir correctament el major nombre de suspensos possible) d'entre tots els models basats en el preprocessament de dades específic que hem realitzat a la primera fase i amb els diferents *kernels* que ofereix *Scikit-Learn*. El camí cap a l'obtenció de models superiors als que presentem passa, doncs, per un preprocessament diferent i per un ús de funcions *kernels* dissenyades *ad hoc*. Cal notar que considerar en el projecte dos preprocessaments, el de 30 variables i el de 10 variables, ens ha portat a millors resultats que si haguéssim descartat un dels dos (doncs per unes certes assignatures el model òptim és de 30 variables i per altres de 10).

Les dues opcions de millora que acabem de comentar són les més senzilles i plausibles. Com en tot problema de *Data Mining*, a banda de manipular les dades existents de manera diferent, també ens podríem plantejar ampliar les dades dels nostre problema (obtenir les notes desglossades dels exàmens parcials, finals i de reavaluació, dades sobre la compaginació de feina i estudis) o la integració de dades que hem decidit no considerar (les notes PAU, el sexe dels estudiants, els codis postals dels centres d'educació secundària i dels llocs de residencia). L'obtenció de dades addicionals és però, en general i en el nostre cas particular, una opció complicada.

Una altra alternativa és per suposat l'ús d'una família diferents d'algorismes de modelatge: descartar els SVM i utilitzar una tècnica diferent de *Machine Learning*. També podríem recórrer a algorismes diferents per complementar els SVM; podríem per exemple preveure amb tècniques de regressió el nombre total d'assignatures suspeses al tercer quadrimestre i introduir-lo (en el cas de tenir un bon model) com a variable X_i als models SVM.

Una manipulació diferent de les dades, un ús de *kernels* astut, l'ampliació de les dades, i l'ús d'altres tècniques de *Machine Learning* són doncs les opcions per a millorar els resultats obtinguts en el projecte.

Resulta difícil valorar aquests resultats obtinguts, perquè són molt desiguals. En el millor dels casos, en l'assignatura de Mecànica, aconseguim preveure la gran majoria de suspensos reals, més de 8 de cada 10, i fer-ho sense massa equivocacions, amb 2 de cada 3 prediccions de suspensos sent correctes. En el pitjor dels casos, però, en l'assignatura de Mètodes numèrics, prediem només 1 de cada 7 suspensos.

D'altra banda, no ens havíem proposat com és lògic objectius en termes numèrics. Destaquem finalment que en les assignatures d'Electromagnetisme, Materials i Mecànica, aconseguim la meitat o més dels suspensos reals, equivocant-nos aproximadament en la meitat de les nostres prediccions.

En conjunt, hem complert els objectius inicials que ens havíem plantejat. Hem executat amb èxit un projecte de *Data Mining* de petita escala, amb totes les fases però d'un projecte d'aquest disciplina i amb una comprensió íntegra de totes elles, del sentit i la fi del preprocessament de dades, del funcionament dels algorismes SVM, i dels passos de les metodologies per escollir un model òptim.

Bibliografia

[1] BATHIA, Parteek. "Introduction to Data Mining". A *Data Mining and Data Warehousing. Principles and Practical Techniques*. Cambridge: Cambridge University Press, 2019, p. 17-27.

[2] BOSER, Bernhard, GUYON, Isabelle i VAPNIK, Vladimir. "A Training Algorithm for Optimal Margin Classifiers". A Haussler, David (ed.), *Proceedings of annual conference on computational learning theory.* Pittsburgh. ACM Press, 1992, p. 144-152.

[3] BRAMER, Max. "Estimating the Predictive Accuracy of a Classifier". A *Principles of Data Mining*. Londres. Springer, 2020, p. 79-92.

[4] CORTES, Corinna, VAPNIK, Vladimir (1995) *Support Vector Networks*. *Mach Learn*, 20 (3), p. 273–297.

[5] GARCÍA, José A. *Líneas de investigación en minería de datos en aplicaciones en ciencia e ingeniería. Estado del arte y perspectivas. ArXiv.* 2016.

[6] HSU, Chih-Wei, *et al.* "A Practical Guide to Support Vector Classification". 2016. [https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf, 15 de juny de 2020].

[7] KANTARDZIC, Mehmed. "Data Mining Concepts". A *Data Mining. Concepts, Models, Methods, and Algorithms*. Hoboken: Wiley, 2020, p. 1-30.

[8] ———. "Common Learning Tasks". A *Data Mining. Concepts, Models, Methods, and Algorithms*. Hoboken: Wiley, 2020, p. 112-117.

[9] ———. "Support Vector Machines". A Data Mining. Concepts, Models, Methods, and Algorithms. Hoboken: Wiley, 2020, p. 117-130.

[10] ———. "Logistic Regression" A Data Mining. Concepts, Models, Methods, and Algorithms. Hoboken: Wiley, 2020, p. 184-185.

[11] LAROSE, Chantal D. & LAROSE, Daniel T. "Introduction to Data Science". A *Data Science using Python and R*. Hoboken: Wiley, 2019, p. 1-9.

[12] LEAVY, Susan. *Gender Bias in Artificial Intelligence. The Need for Diversity and Gender Theory in Machine Learning*. *ACM/IEEE 1st International Workshop on Gender Equality in Software Engineering*. 2018.

[13] ROGEL-SALAZAR, Jesús. "The Machine that Goes 'Ping': Machine Learning and Pattern

Recognition". A *Data Science and Analytics with Python*. Boca Raton: CRC, 2017, p. 87-131.

[14] ———. "Unicorns and Horses. Classification". A *Data Science and Analytics with Python*. Boca Raton: CRC, 2017, p. 195-241.

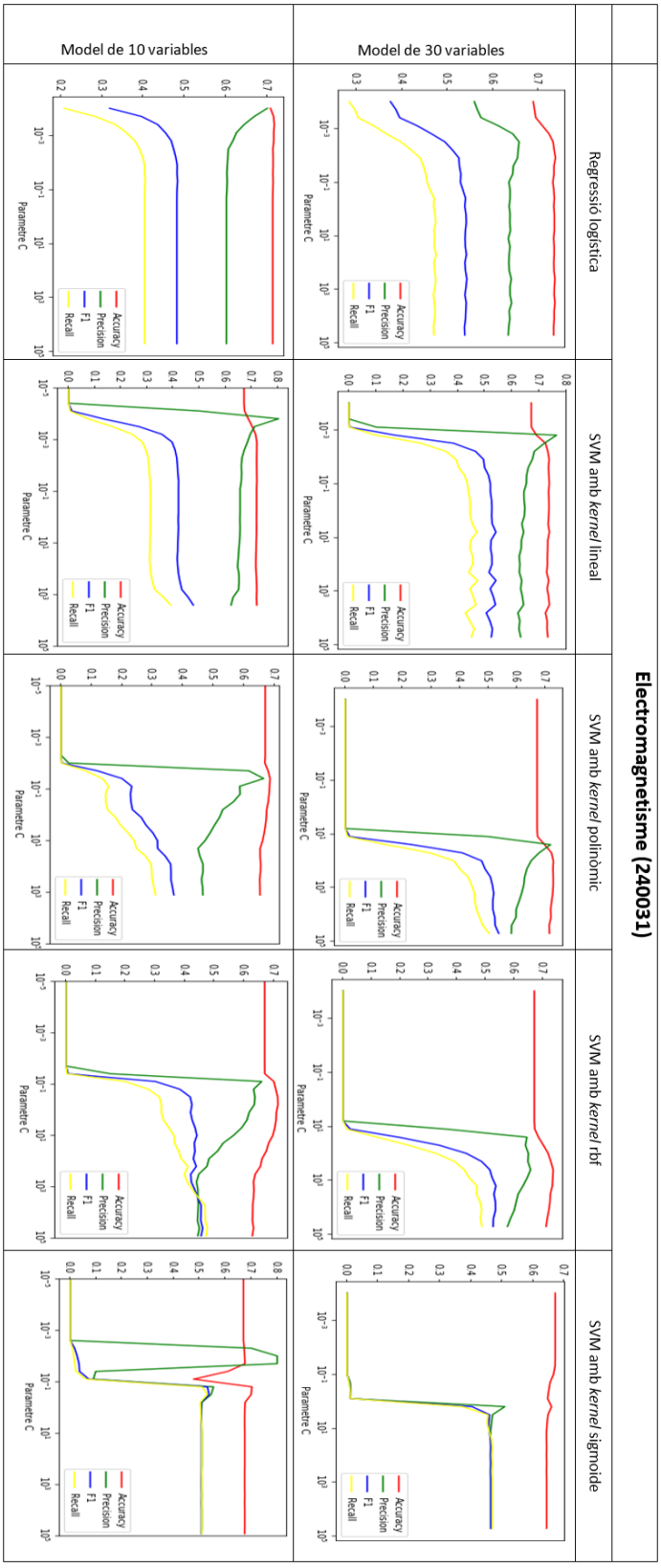
[15] ———. "Less is more. Dimensionality Reduction". A *Data Science and Analytics with Python*. Boca Raton: CRC, 2017, p. 285-326.

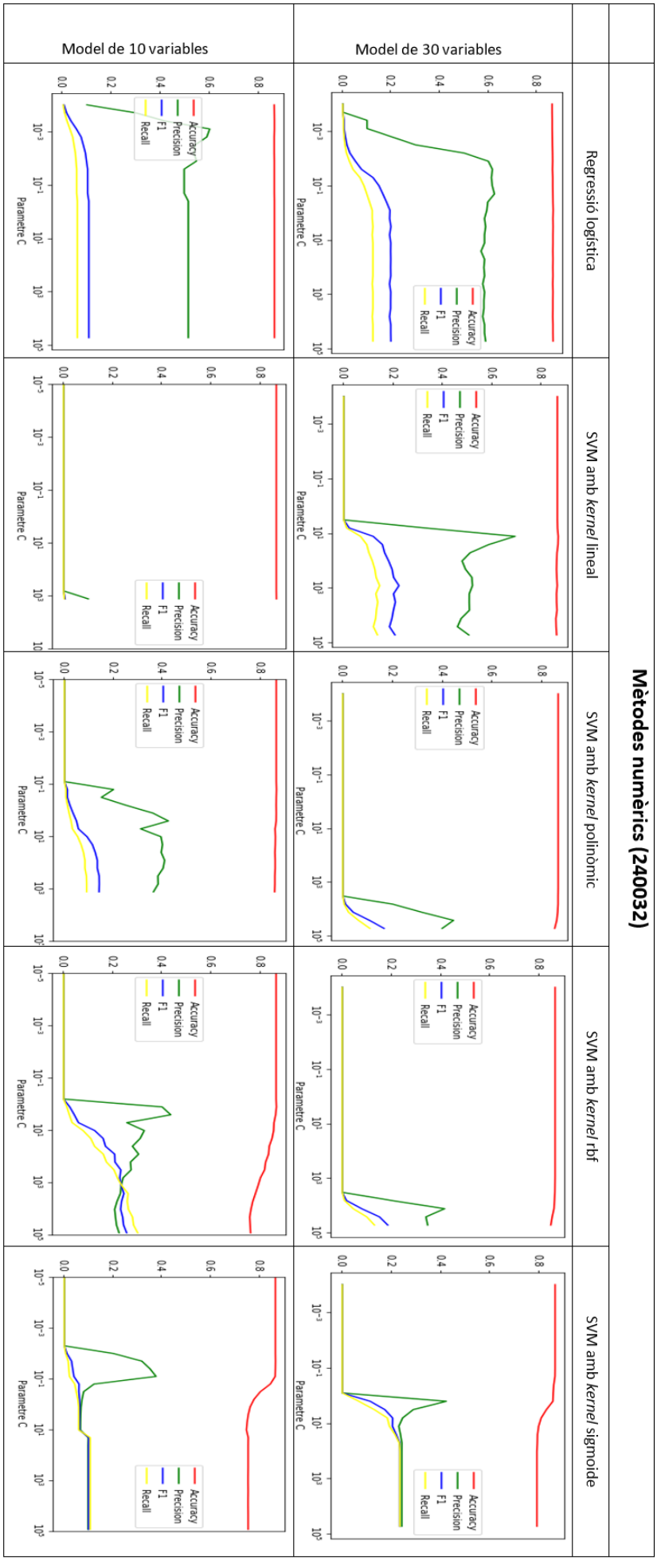
[16] ZHANG, Xinhua. "Support Vector Machines". A Sammut, Claude i Webb, Geoffrey I. (eds.), *Encyclopedia of Data Mining and Machine Learning*. Nova York: Springer, 2017, p. 1214-1219.

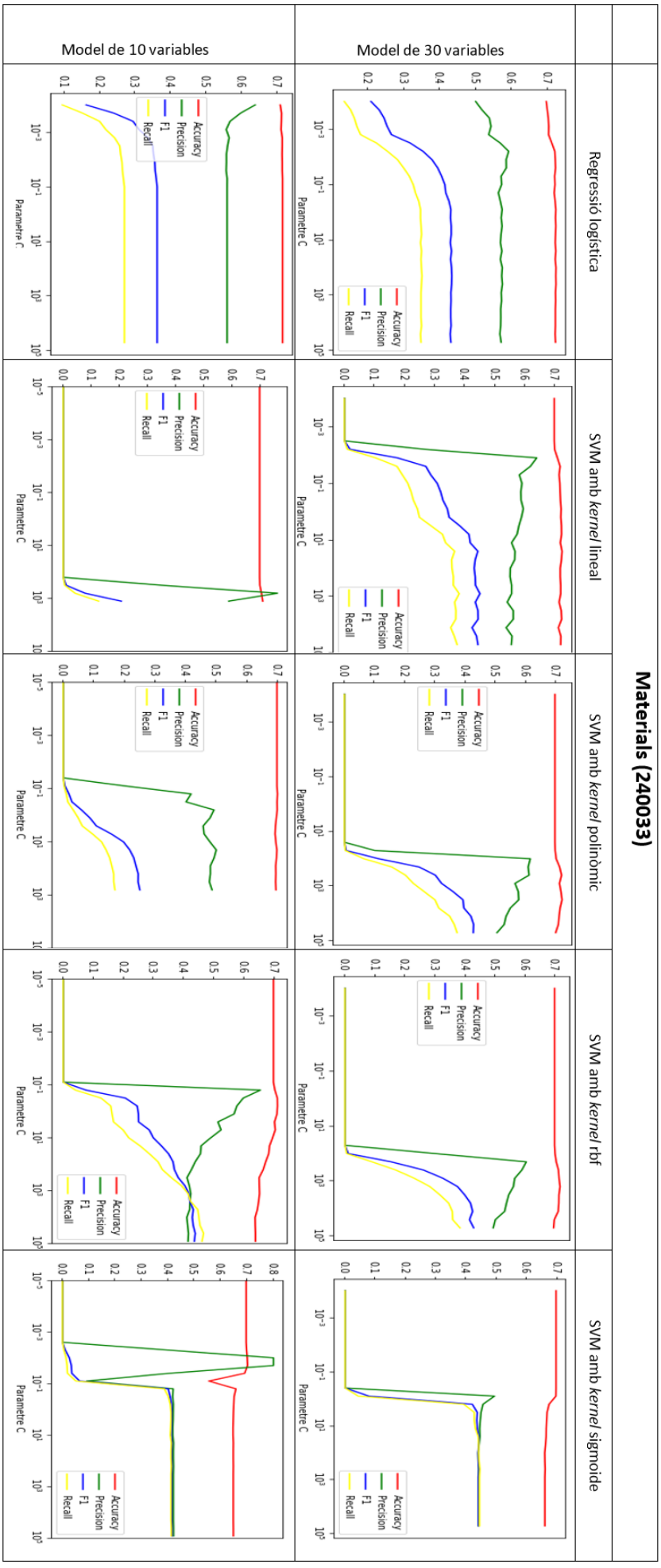
[17] Documentació oficial de la llibreria *Pandas*, "Reshaping". 2020. [https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html, 2 de febrer de 2020].

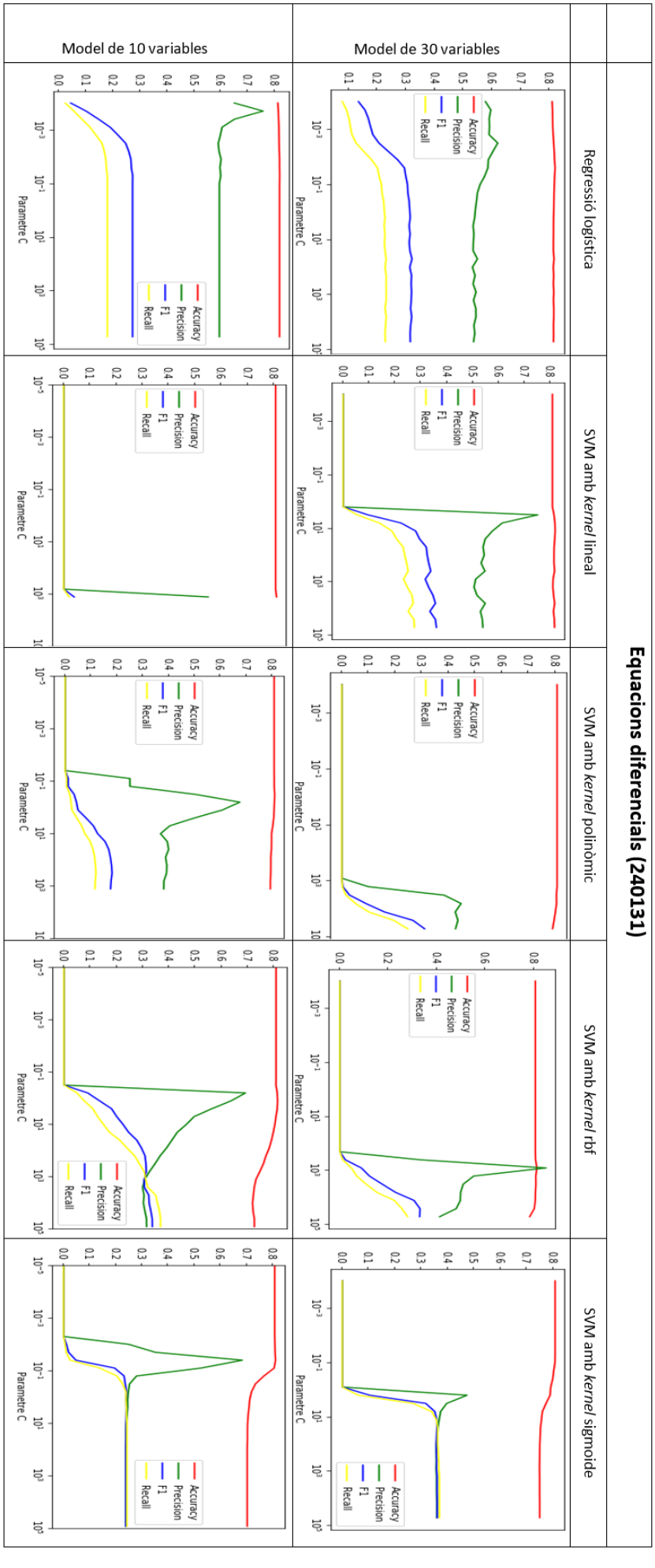
[18] Usuari de GitHub "WittmannF". "compare-svm-kernels.py". 2016. [https://gist.github.com/WittmannF/60680723ed8dd0cb993051a7448f7805, 9 de maig de 2020].

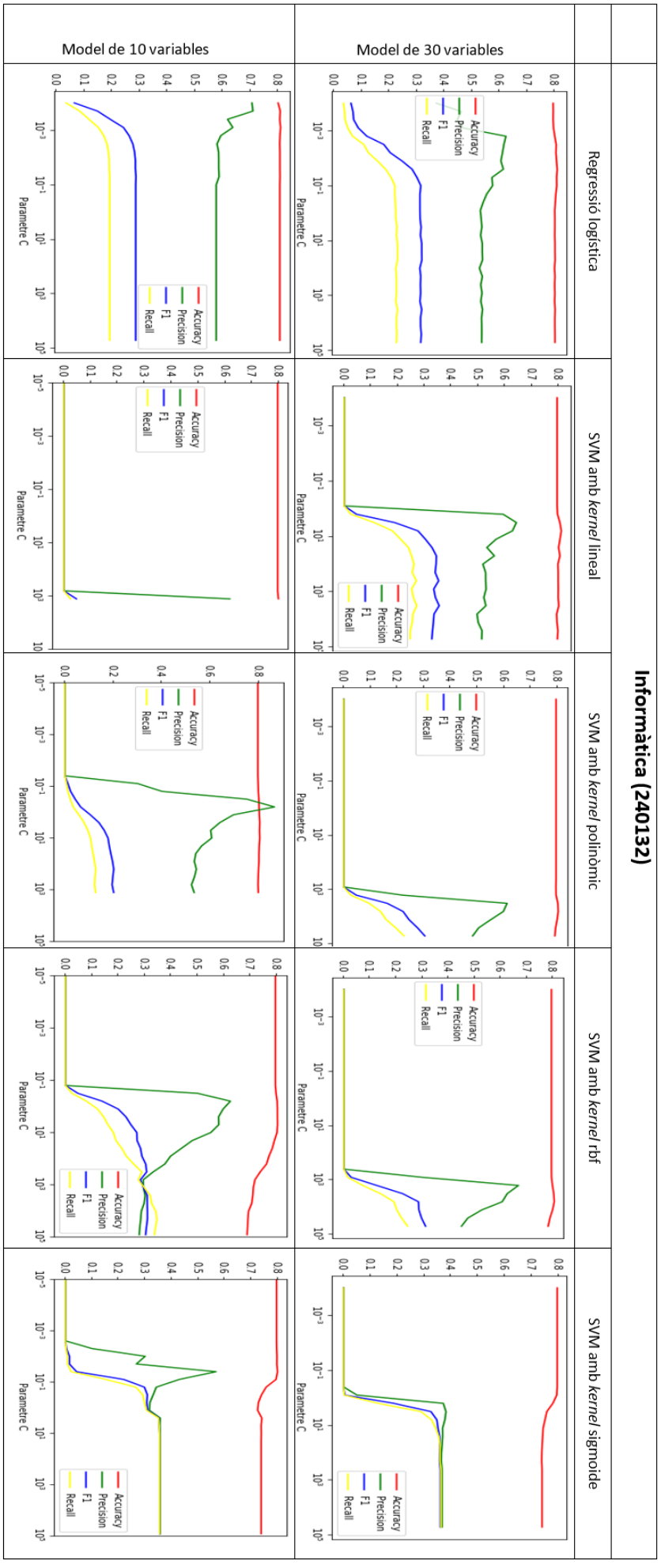
7. Annex 1: Gràfics de les mètriques dels models

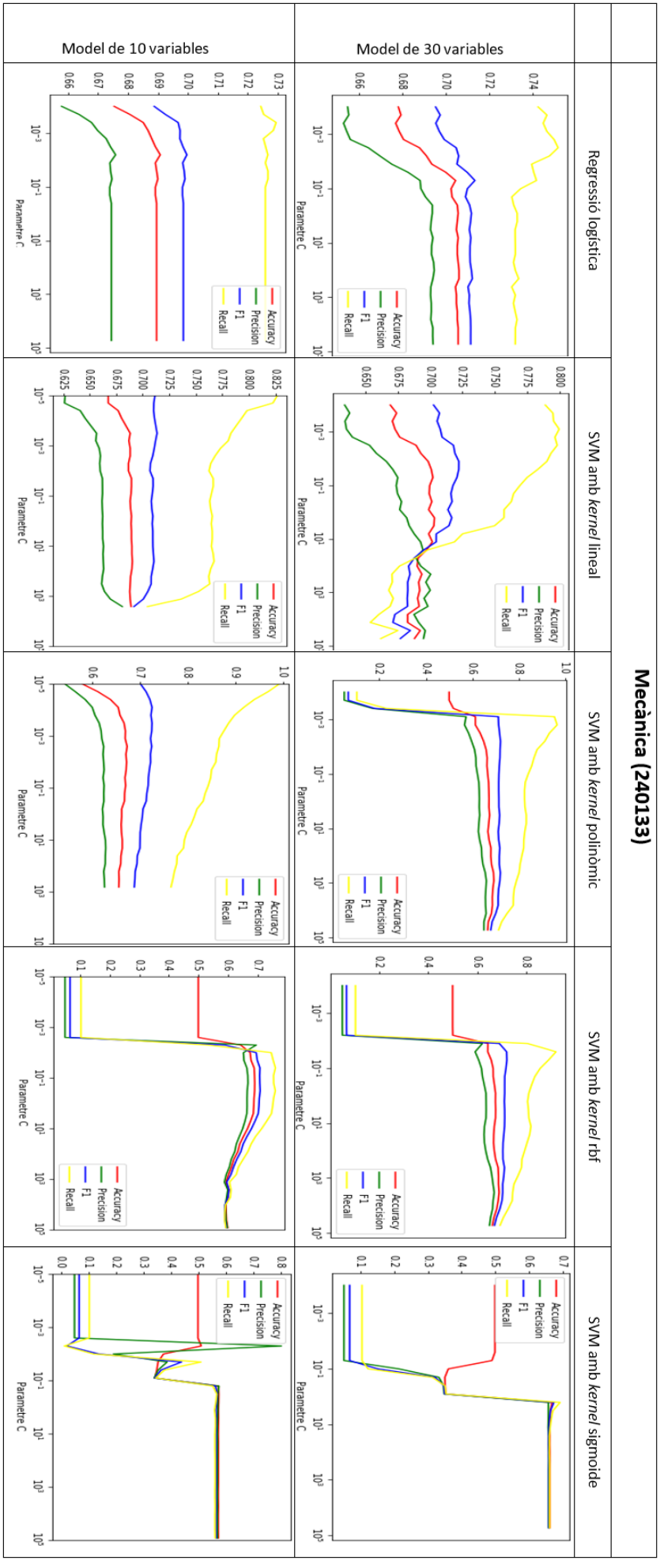



Mètodes numèrics (240032)
Model de 30 variables
Model de 10 variables
Regressió logística
SVM amb kernel lineal
SVM amb kernel polinòmic
SVM amb kernel rbf
SVM amb kernel sigmoide
Accuracy
Precision
F1
Recall
Parametre C


Materials (240033)
Model de 30 variables
Model de 10 variables
Regressió logística
SVM amb kernel lineal
SVM amb kernel polinòmic
SVM amb kernel rbf
SVM amb kernel sigmoide
Accuracy
Precision
F1
Recall
Parametre C


Equacions diferencials (240131)
Regressió logística
SVM amb kernel lineal
SVM amb kernel polinòmic
SVM amb kernel rbf
SVM amb kernel sigmoide
Model de 30 variables
Model de 10 variables
Accuracy
Precision
F1
Recall
Parametre C


Informàtica (240132)
Regressió logística
SVM amb kernel lineal
SVM amb kernel polinòmic
SVM amb kernel rbf
SVM amb kernel sigmoide
Model de 30 variables
Model de 10 variables
Accuracy
Precision
F1
Recall
Parametre C


Mecànica (240133)
Model de 30 variables
Model de 10 variables
Regressió logística
SVM amb kernel lineal
SVM amb kernel polinòmic
SVM amb kernel rbf
SVM amb kernel sigmoide
Accuracy
Precision
F1
Recall
Parametre C

8. Annex 2: Codi del preprocessament

```python
import numpy as np
import pandas as pd
from time import process_time

t1_start = process_time()

def PercMitja (row, MitgesConv):
    index=MitgesConv[ (MitgesConv['CURS']==row['CURS']) & (MitgesConv['CODI_UPC_UD']==row['CODI_UPC_UD']) & (MitgesConv['QUAD']==row['QUAD']) ].index
    mitja=MitgesConv.loc[index, 'NOTA_NUM_DEF']

    Percentatge=int((row['NOTA_NUM_DEF']/mitja*100))
    return Percentatge


#Carreguem dades Fase Inicial
FaseIni=pd.read_excel('/Users/rn/Google Drive/2019-20/TFG Etseib/Gener/qfaseini.xlsx', sheet_name='Dades_Solano_qualif')

#Treiem expedients que no siguin de Tecnologies industrials (que te per codi 752)
Index = FaseIni[ FaseIni['CODI_PROGRAMA'] != 752 ].index
FaseIni.drop(Index, inplace=True)

#Treiem les convocatories sense nota
#i les que tenen un 0 (si no han fet res de l assignatura millor no considerarlos)
FaseIni.dropna(subset=['NOTA_NUM_DEF'], inplace=True)

Index = FaseIni[ FaseIni['NOTA_NUM_DEF'] == 0 ].index
FaseIni.drop(Index, inplace=True)

#Treiem els estudiants del grup "CONV"
Index = FaseIni[ FaseIni['GRUP_CLASSE'] == 'CONV' ].index
FaseIni.drop(Index, inplace=True)

#Treiem els estudiants de quadrimestre 0
Index = FaseIni[ FaseIni['QUAD'] == 0].index
FaseIni.drop(Index, inplace=True)

#------------------------
#Afegim una columna 'MITJA_CONV' que sigui la mitja de l assignatura en aquella convocatoria
#------------------------

#ITERANT
#for i in range (2010, 2018):
#    for assignatura in chain (range (240011, 240016), range (240021, 240026)):
#        for quadrimestre in range (1, 3):
#            aux=FaseIni[(FaseIni['CURS']==i) & (FaseIni['CODI_UPC_UD']==assignatura) & (FaseIni['QUAD']==quadrimestre)]
#            mitja=aux['NOTA_NUM_DEF'].mean()
#
#            aux2=aux[(aux['NOTA_NUM_DEF']<5)]
#            if len(aux.index)!=0:
#                percentatge=(len(aux2.index)/len(aux.index))*100
#
#            Index = FaseIni[(FaseIni['CURS']==i) & (FaseIni['CODI_UPC_UD']==assignatura) & (FaseIni['QUAD']==quadrimestre)].index
#            FaseIni.loc[Index, 'MITJA_CONV']=mitja
#            FaseIni.loc[Index, '%SUSPESOS_CONV']=percentatge

#SENSE INTERAR (groupby+apply)
#Utilitzo Groupby per obtenir un DF amb mitges de convocatories
MitgesConv=FaseIni[['CURS', 'CODI_UPC_UD', 'QUAD', 'NOTA_NUM_DEF']].groupby(['CURS', 'CODI_UPC_UD', 'QUAD'], as_index=False).mean()


```

```
61
62 #Afegeixo columna %Mitja que es el tant per cent de la nota del alumne respecte a la mitja
63 FaseIni['%Mitja']=FaseIni.apply(lambda x: PercMitja(x, MitgesConv), axis=1)
64
65 #-----------------------------------------------------
66 #Ara ja hem obtingut les columnes adicionals que volem
67 #Fem pivoting per tenir el DF per expedients i no per convocatories
68
69 #Hem de tractar les assignatures repetides per poder fer pivoting
70 #-----------------------------------------------------
71
72 #ITERANT
73 #El que farem sera canviar la fila de la convocatoria repetida
74 #canviant el 'CODI_UPC_UD' afegint un 0 per primera repeticio
75 #dos 0 per segona
76 #tres 0 per tercera
77 #
78 #Repeticions=FaseIni[FaseIni.duplicated(['CODI_EXPEDIENT', 'CODI_UPC_UD'], keep=False)]
79 #
80 #while not Repeticions.empty:
81 #
82 #    cont=0
83 #    for index, row in Repeticions[(Repeticions['CODI_EXPEDIENT']==Repeticions['CODI_EXPEDIENT'].iloc[0]) & (Repeticions['CODI_UPC_UD']==Repeticions['CODI_UPC_UD'].iloc[0])].iterrows():
84 #        cont+=1
85 #        if cont==2:
86 #            #1a repeticio
87 #            FaseIni['CODI_UPC_UD'].loc[index]=FaseIni['CODI_UPC_UD'].loc[index]*10
88 #
89 #        if cont==3:
90 #            #2a repeticio
91 #            FaseIni['CODI_UPC_UD'].loc[index]=FaseIni['CODI_UPC_UD'].loc[index]*100
92 #
93 #        if cont==4:
94 #            FaseIni['CODI_UPC_UD'].loc[index]=FaseIni['CODI_UPC_UD'].loc[index]*1000
95 #
96 #        if cont==5:
97 #            FaseIni['CODI_UPC_UD'].loc[index]=FaseIni['CODI_UPC_UD'].loc[index]*10000
98 #
99 #
100 #    IndexEliminar=Repeticions[(Repeticions['CODI_EXPEDIENT']==Repeticions['CODI_EXPEDIENT'].iloc[0]) & (Repeticions['CODI_UPC_UD']==Repeticions['CODI_UPC_UD'].iloc[0])].index
101 #    Repeticions.drop(IndexEliminar, inplace=True)
102
103
104 #SENSE ITERAR
105 #Utilitzo groupby per obtenir DF amb Expedient, Assignatura, vegades cursada
106 Repeticions=FaseIni[['CODI_EXPEDIENT', 'CODI_UPC_UD',]].groupby(['CODI_EXPEDIENT', 'CODI_UPC_UD',], as_index=False).size()
107 Repeticions=Repeticions.to_frame()
108 Repeticions=pd.DataFrame(Repeticions.to_records())
109 Repeticions.columns=['CODI_EXPEDIENT', 'CODI_UPC_UD', 'Rep']
110
111
112 #Elimino les convocatories no aprovades
113 Index = FaseIni[ FaseIni['NOTA_NUM_DEF'] < 5 ].index
114 FaseIni.drop(Index, inplace=True)
115
116 #Afegeixo columna 'Rep' amb el nombre de vegades que l estudiant ha cursat l assignatura
117 FaseIni=pd.merge(FaseIni, Repeticions, how='left', on=['CODI_EXPEDIENT', 'CODI_UPC_UD'])
118
119
120 #Pivoting
121 FaseIniExp=FaseIni.pivot(index= 'CODI_EXPEDIENT', columns='CODI_UPC_UD', values=['NOTA_NUM_DEF', '%Mitja', 'Rep'])
122
123 #Trec estudiants que tenen NaN a alguna assignatura (que vol dir que no l han aprovat i que no han passat la FI)
124 FaseIniExp.dropna(inplace=True)
125
126 t1_stop = process_time()
127 print("Elapsed time during the whole program in seconds:",  t1_stop-t1_start)
128
129 FaseIniExp['NOTA_NUM_DEF'].to_csv("Dades tractadesNota.csv")
130 FaseIniExp['%Mitja'].to_csv("Dades tractadesMitges.csv")
131 FaseIniExp['Rep'].to_csv("Dades tractadesRep.csv")
132
```

9. Annex 3: Codi del modelatge

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
from sklearn import svm


#Carregar Dades
DadesXNotes=pd.read_csv('Dades tractadesNota.csv')
DadesXMitges=pd.read_csv('Dades tractadesMitges.csv')
DadesXRep=pd.read_csv('Dades tractadesRep.csv')
DadesY=pd.read_excel('Dades tractades No Ini20.xlsx', sheet_name='Sheet1')

Dades=pd.merge(DadesXNotes, DadesXMitges, how='left', on='CODI_EXPEDIENT')
Dades=pd.merge(Dades, DadesXRep, how='left', on='CODI_EXPEDIENT')
Dades=pd.merge(Dades, DadesY, how='left', on='CODI_EXPEDIENT')

Dades.dropna(inplace=True)

X=np.array(Dades.loc[:,'240011':'240025R']) #(Aquest codi es pels models de 30 variables)

for a in ['240031', '240032', '240033', '240131', '240132', '240133']:
    for k in ['linear', 'poly', 'rbf', 'sigmoid']:
        Y=np.array(Dades[a])
        Resultats=pd.DataFrame(columns=['Accuracy','Precision', 'F1', 'Recall'])

        c=0.0001
        while c<100001:
            #Fixem Classifier que utilitzarem
            Clf=svm.SVC(kernel=k, C=c)

            #Evaluem/validem model (cross validation)
            seed=5
            kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
            scores=['accuracy', 'precision', 'recall', 'f1']

            cv_results = model_selection.cross_validate(Clf, X, Y, cv=kfold, scoring=scores)

            #Guardem resultats
            Resultats.loc[c, ['Accuracy']]=cv_results['test_accuracy'].mean()
            Resultats.loc[c, ['Precision']]=cv_results['test_precision'].mean()
            Resultats.loc[c, ['F1']]=cv_results['test_f1'].mean()
            Resultats.loc[c, ['Recall']]=cv_results['test_recall'].mean()
```

```
46
47          #
48          #    print ('Matriu de confusio:')
49          #    kf = model_selection.KFold(n_splits=10, random_state=seed)
50          #    MC=np.array([[0, 0], [0, 0]])
51          #
52          #
53          #    for train_index, test_index in kf.split(X):
54          #
55          #       X_train, X_test = X[train_index], X[test_index]
56          #       Y_train, Y_test = Y[train_index], Y[test_index]
57          #
58          #       Clf.fit(X_train, Y_train)
59          #       MC=MC+confusion_matrix(Y_test, Clf.predict(X_test))
60          #
61          #    print (MC)
62
63              c=c*2
64
65          #Generem excel
66          temp=k+a+'.xlsx'
67          Resultats.to_excel(temp)
68
69          #Generem grafic
70          Resultats=Resultats.reset_index()
71          ax=plt.gca()
72          Resultats.plot(kind='line', x='index', y='Accuracy', color='red', ax=ax)
73          Resultats.plot(kind='line', x='index', y='Precision', color='green', ax=ax)
74          Resultats.plot(kind='line', x='index', y='F1', color='blue', ax=ax)
75          Resultats.plot(kind='line', x='index', y='Recall', color='yellow', ax=ax)
76          plt.xlabel('Parametre C')
77          plt.xscale('log')
78
79          temp2=k+a+'.png'
80          plt.savefig(temp2)
81          plt.clf()
```